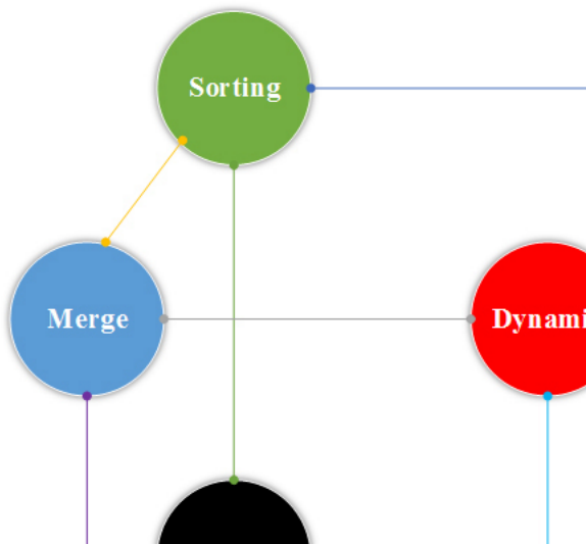
Algorit

Java



Algorithms Java

YANG HU

Simple is the beginning of wisdom. From the essence of practice,this book to briefly explain the concept and vividly cultivate programming interest, you will learn it easy, fast and well.

http://en.verejava.com

Copyright © 2020 Yang Hu

All rights reserved.

ISBN : 9798657092882

CONTENTS

1 . Linear Table Definition
2 . Maximum Value
3. Bubble Sorting Algorithm
4 . Minimum Value
5 . Select Sorting Algorithm
6. Linear Table Append
7. Linear Table Insert
8. Linear Table Delete

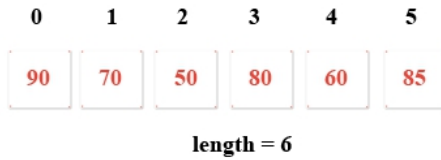
9 . Insert Sorting Algorithm
10 . Reverse Array
11. Linear Table Search
12. Dichotomy Binary Search
13 . Shell Sorting
14. Single Link
    14.1 Create and Initialization
    14.2 Add Node
    14.3 Insert Node
    14.4 Delete Node
15. Doubly Linked List
    15.1 Create and Initialization
    15.2 Add Node
    15.3 Insert Node
    15.4 Delete Node
16. One-way Circular LinkedList
    16.1 Initialization and Traversal
    16.2 Insert Node
    16.3 Delete Node
17. Two-way Circular LinkedList
    17.1 Initialization and Traversal
    17.2 Insert Node
    17.3 Delete Node
18. Queue
19. Stack
20. Recursive Algorithm
21 . Quick Sorting
22. Two-way Merge Algorithm
23. Binary Search Tree
    23.1 Construct a binary search tree
    23.2 Binary search tree In-order traversal

23.3 Binary search tree Pre-order traversal
23.4 Binary search tree Post-order traversal
23.5 Binary search tree Maximum and minimum
23.6 Binary search tree Delete Node
24. Binary Heap Sorting
25. Hash Table
26. Graph
26.1 Directed Graph and Depth-First Search
26.2 Directed Graph and Breadth-First Search
26.3 Directed Graph Topological Sorting
27 . Towers of Hanoi
28 . Fibonacci
29 . Dijkstra
30 . Mouse Walking Maze
31 . Eight Coins
32 . Knapsack Problem
33 . Josephus Problem

Linear Table Definition

Linear Table:
Sequence of elements, is a one-dimensional array.

1 . Define a one-dimensional array of student scores



TestOneArray.java

```
public class TestOneArray {
    public static void main(String[] args) {
        int [] scores = { 90, 70, 50, 80, 60, 85 };
        for (int i = 0; i < scores.length ; i++) {
            System.out .print(scores[i] + "," );
        }
    }
}
```

Result:

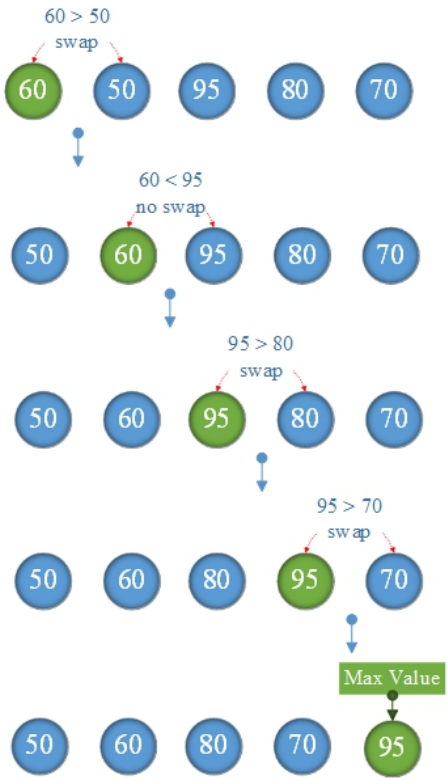
90, 70, 50, 80, 60, 85

Maximum Value

Maximum of Integer Sequences:

1 . Algorithmic ideas

Compare arrays[i] with arrays[i + 1] , if arrays[i] > arrays[i + 1] are exchanged. So continue until the last number, arrays[length - 1] is the maximum.



TestMaxValue.java

```
public class TestMaxValue {
    public static void main(String[] args) {
        int [] scores = { 60, 50, 95, 80, 70 };
        int maxValue = max (scores);
        System.out .println("Max Value = " + maxValue);
    }
    public static int max(int [] arrays) {
        // Maximum initialization value is 0
        for (int i = 0; i < arrays.length - 1; i++) {
            if (arrays[i] > arrays[i + 1]) { // swap
                int temp = arrays[i];
                arrays[i] = arrays[i + 1];
                arrays[i + 1] = temp;
            }
        }
        int maxValue = arrays[arrays.length - 1];
        return maxValue;
    }
}
```

Result:

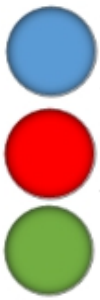
```
Max Value = 95
```

Bubble Sorting Algorithm

Bubble Sorting Algorithm:
Compare arrays[j] with arrays[j + 1] , if arrays[j] > arrays[j + 1] are exchanged.
Remaining elements repeat this process, until sorting is completed.

Sort the following numbers from small to large

Explanation:

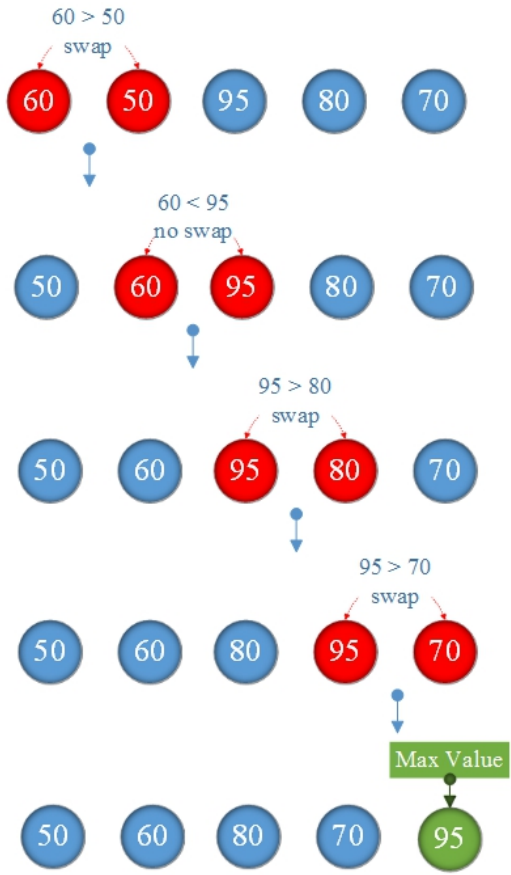


No sorting,

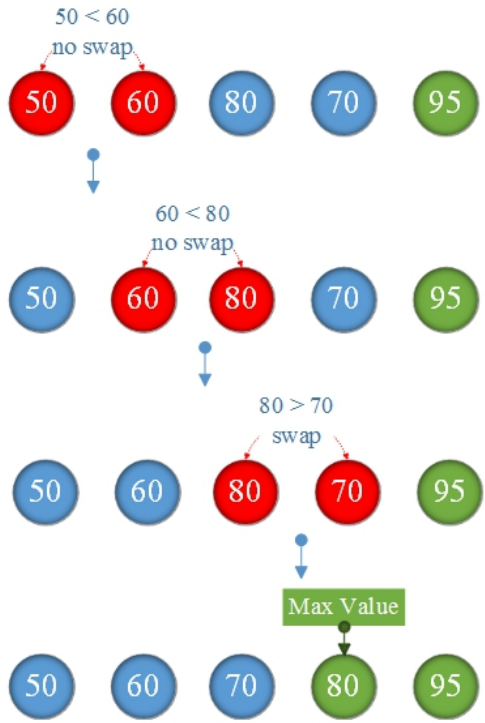
Comparing,

Already sorted

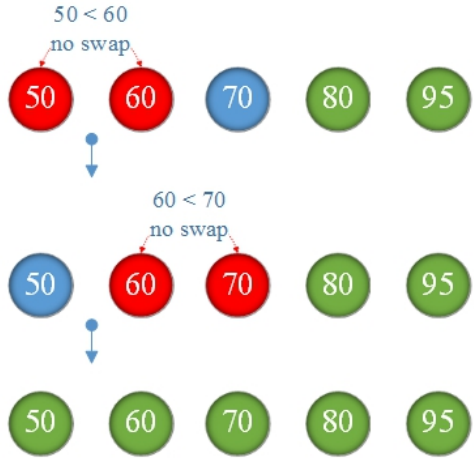
1 . First sorting:



2 . Second sorting:



3 . Third sorting:



No swap so terminate sorting : we can get the sorting numbers from small to large

TestBubbleSort.java

```
public class TestBubbleSort {
    public static void main(String[] args) {
        int [] scores = { 60, 50, 95, 80, 70 };
        sort (scores);
        for (int i = 0; i < scores.length ; i++) {
            System.out .print(scores[i] + "," );
        }
    }
    public static void sort(int [] arrays) {
        for (int i = 0; i < arrays.length - 1; i++) {
            boolean isSwap = false ;
            for (int j = 0; j < arrays.length - i - 1; j++) {
                if (arrays[j] > arrays[j + 1]) {//swap
                    int temp = arrays[j];
                    arrays[j] = arrays[j + 1];
                    arrays[j + 1] = temp;
                    isSwap = true ;
                }
            }
            if (!isSwap) //No swap so terminate sorting
            {
                break ;
            }
        }
    }
}
```

Result:

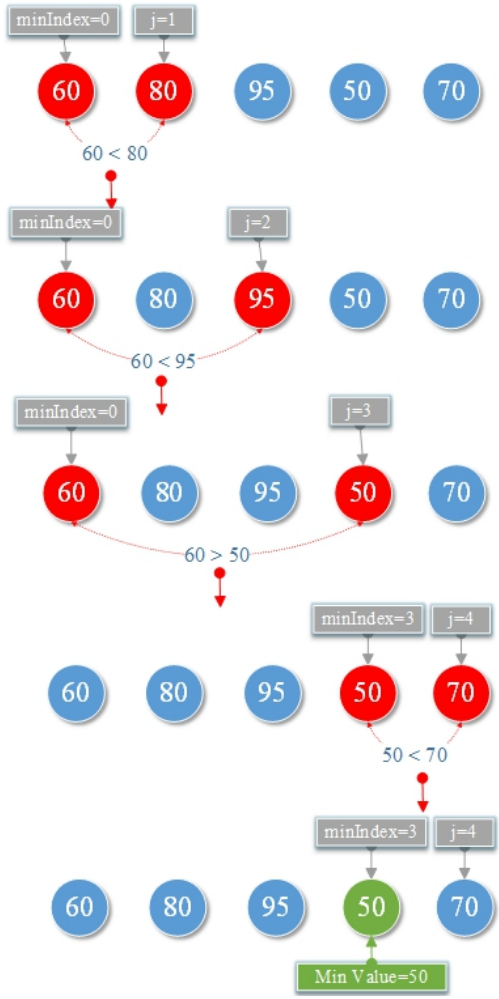
50,60,70,80,95,

Minimum Value

Search the Minimum of Integer Sequences:

1 . Algorithmic ideas

Initial value minIndex = 0, j = 1 Compare arrays[minIndex] with arrays[j]
if arrays[minIndex] > arrays[j] then minIndex = j, j++ else j++.
continue until the last number, arrays[minIndex] is the Min Value.


minIndex=0
j=1
60
80
95
50
70
60 < 80
minIndex=0
j=2
60
80
95
50
70
60 < 95
minIndex=0
j=3
60
80
95
50
70
60 > 50
minIndex=3
j=4
60
80
95
50
70
50 < 70
minIndex=3
j=4
60
80
95
50
70
Min Value=50

TestMinValue.java

```
public class TestMinValue {
    public static void main(String[] args) {
        int [] scores = { 60, 80, 95, 50, 70 };
        int minValue = min (scores);
        System.out .println("Min Value = " + minValue);
    }
    public static int min(int [] arrays) {
        int minIndex = 0;// the index of the minimum
        for (int j = 1; j < arrays.length ; j++) {
            if (arrays[minIndex] > arrays[j]) {
                minIndex = j;
            }
        }
        return arrays[minIndex];
    }
}
```

Result:

```
Min Value = 50
```

Select Sorting Algorithm

Select Sorting Algorithm:
Sorts an array by repeatedly finding the minimum element from unsorted part and putting it at the beginning.

Sort the following numbers from small to large

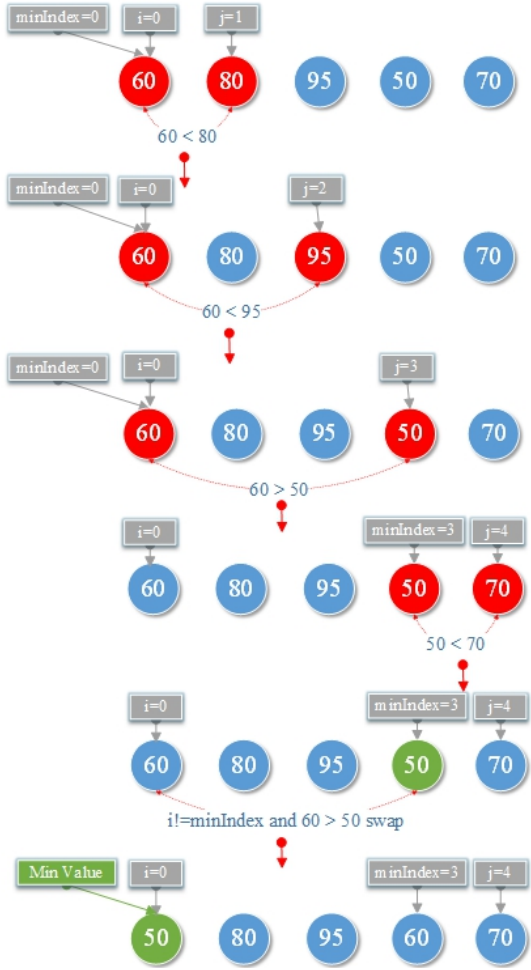
Explanation:

No sorting,

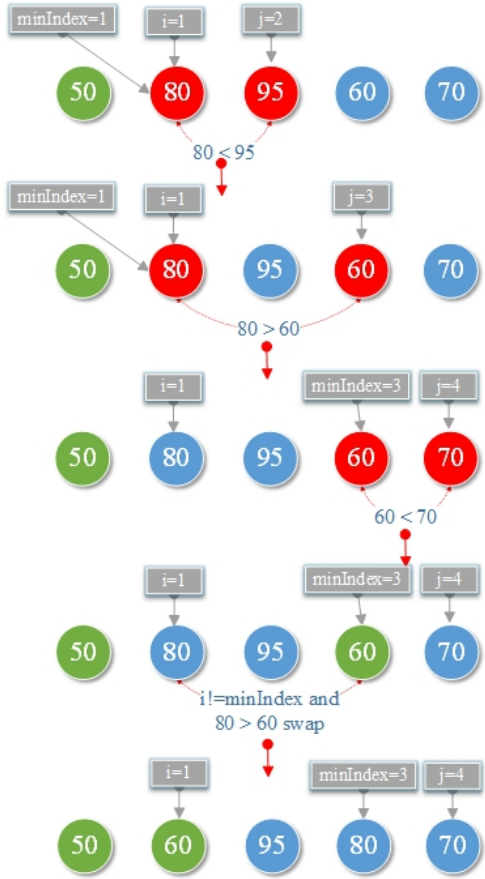
Comparing,

Already sorted.

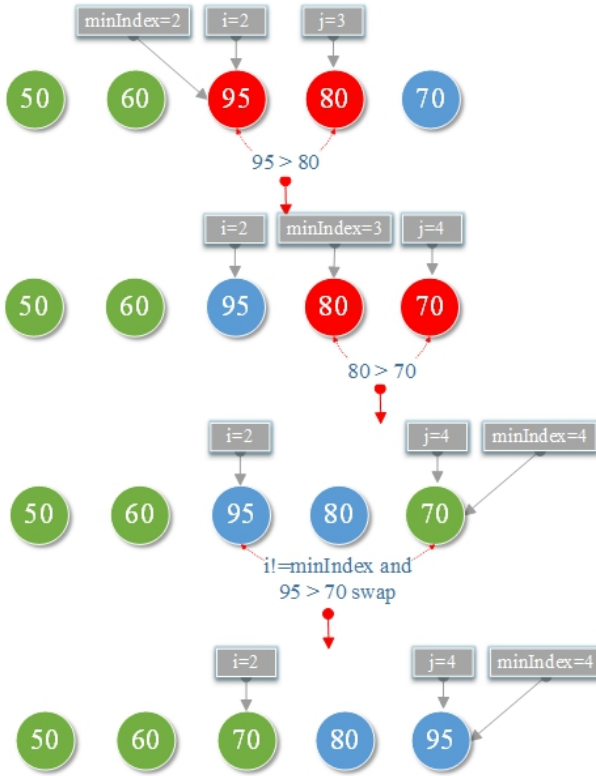
1 . First sorting:



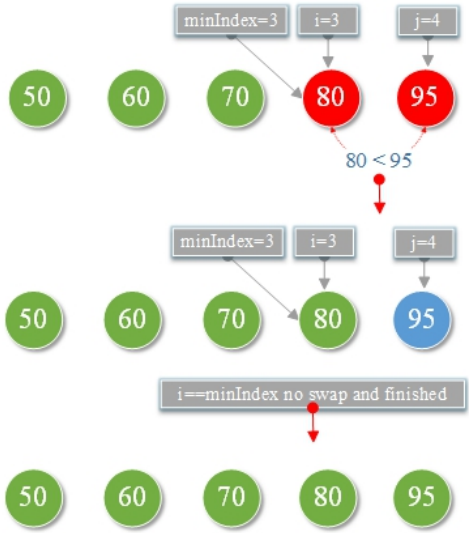
2 . Second sorting:



3 . Third sorting:



4 . Forth sorting:



we can get the sorting numbers from small to large

TestSelectSort.java

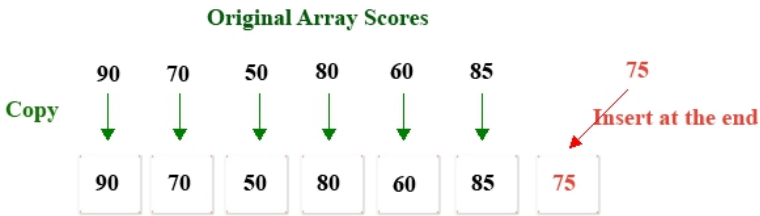
```
public class TestSelectSort {
    public static void main(String[] args) {
        int [] scores = { 60, 80, 95, 50, 70 };
        sort (scores);
        for (int score : scores) {
            System.out .print(score + "," );
        }
    }
    public static void sort(int [] arrays) {
        for (int i = 0; i < arrays.length - 1; i++) {
            int minIndex = i; //  the index of the selected minimum
            for (int j = i + 1; j < arrays.length ; j++) {
                if (arrays[minIndex] > arrays[j]) {
                    minIndex = j;
                }
            }
            if (i != minIndex) //minimum arrays[i] is swaped with
the arrays[minIndex]
            {
                int temp = arrays[i];
                arrays[i] = arrays[minIndex];
                arrays[minIndex] = temp;
            }
        }
    }
}
```

Result:

50,60,70,80,95,

Linear Table Append

1 . Add a score 75 to the end of the one-dimensional array scores.



Analysis:

1. First create a temporary array(tempArray) larger than the original scores array length
2. Copy each value of the scores to tempArray
3. Assign 75 to the last index position of tempArray
4. Finally assign the tempArray pointer reference to the original scores;

TestOneArrayAppend.java

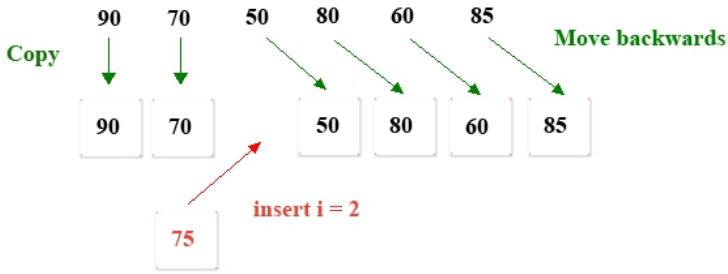
```
public class TestOneArrayAppend {
    public static int [] append(int [] array, int value) {
      //create a new array, length = array.length + 1
        int [] tempArray = new int [array.length + 1];
        for (int i = 0; i < array.length ; i++) {
            tempArray[i] = array[i];
        }
        tempArray[array.length ] = value
        return tempArray;
    }
    public static void main(String[] args) {
        int [] scores = { 90, 70, 50, 80, 60, 85 };
        scores = append (scores, 75);
        for (int i = 0; i < scores.length ; i++) {
            System.out .print(scores[i] + "," );
        }
    }
}
```

Result:

90, 70, 50, 80, 60, 85, 75,

Linear Table Insert

1 . Insert a student's score anywhere in the one-dimensional array scores.



Analysis:

1. First create a temporary array tempArray larger than the original scores array length
2. Copy each value of the previous value of the scores array from the beginning to the insertion position to tempArray
3. Move the scores array from the insertion position to each value of the last element and move it back to tempArray
4. Then insert the score 75 to the index of the tempArray .
5. Finally assign the tempArray pointer reference to the scores;

TestOneArrayInsert.java

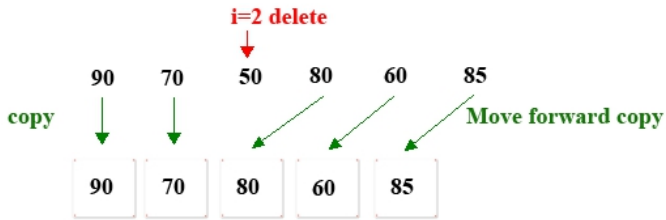
```
public class TestOneArrayInsert {
    public static int [] insert(int [] array, int score, int
insertIndex) {
        int [] tempArray = new int [array.length + 1];
        for (int i = 0; i < array.length ; i++) {
            if (i < insertIndex) {
                tempArray[i] = array[i];
            } else {
                tempArray[i + 1] = array[i];
            }
        }
        tempArray[insertIndex] = score;
        return tempArray;
    }
    public static void main(String[] args) {
        int [] scores = { 90, 70, 50, 80, 60, 85 };
        scores = insert (scores, 75, 2); //Insert 75 into the position:
index = 2
        for (int i = 0; i < scores.length ; i++) {
            System.out .print(scores[i] + "," );
        }
    }
}
```

Result:

90,70,75,50, 80, 60, 85,

Linear Table Delete

1 . Delete the value of the index = 2 from scores array



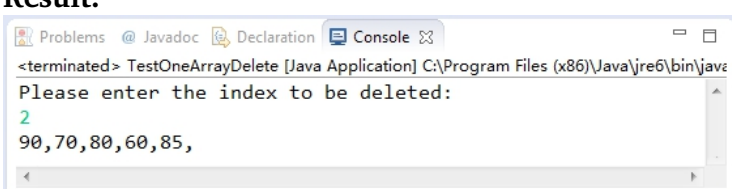
Analysis:

1. Create a temporary array tempArray that length smaller than scores by 1.
2. Copy the data in front of i = 2 to the front of tempArray
3. Copy the array after i = 2 to the end of tempArray
4. Assign the tempArray pointer reference to the scores
5. Printout scores

TestOneArrayDelete.java

```
import java.util.Scanner;
public class TestOneArrayDelete {
    public static int [] delete(int [] array, int index){
     // create a new array, length = array.length - 1
        int [] tempArray = new int [array.length - 1];
        for (int i = 0; i < array.length ; i++) {
            if (i < index) // Copy the data in front of index to the
front of tempArray
                tempArray[i] = array[i];
            if (i > index)  // Copy the array after index to the end of
tempArray
                tempArray[i - 1] = array[i];
        }
        return tempArray;
    }
    public static void main(String[] args) {
        int [] scores = { 90, 70, 50, 80, 60, 85 };
        System.out .println("Please enter the index to be deleted:" );
        Scanner in = new Scanner(System.in );
        int index = in.nextInt();
        scores = delete (scores, index);
        for (int i = 0; i < scores.length ; i++) {
            System.out .print(scores[i] + "," );
        }
    }
}
```

Result:



Insert Sorting Algorithm

Insert Sorting Algorithm:
Take an unsorted new element in the array, compare it with the already sorted element before, if the element is smaller than the sorted element, insert new element to the right position.

Sort the following numbers from small to large

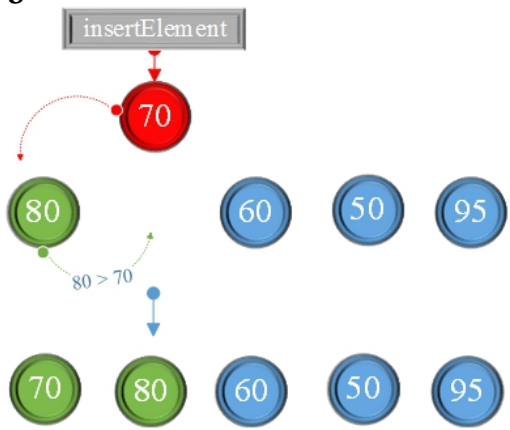
Explanation:

No sorting,

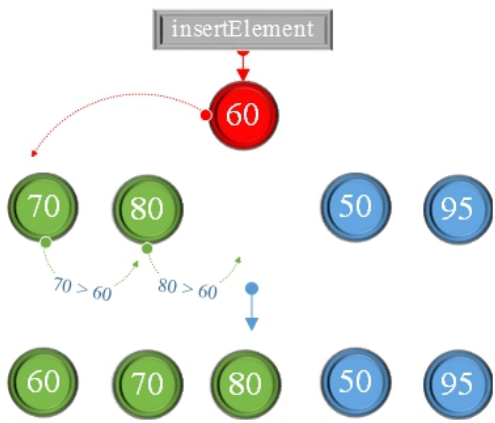
Inserting,

Already sorted

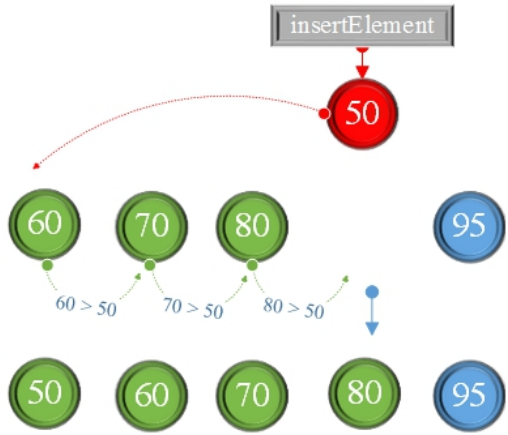
1 . First sorting:



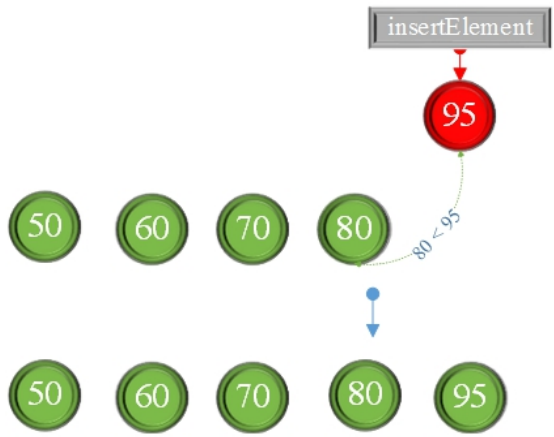
2 . Second sorting:



3 . Third sorting:



4 Third sorting:



TestInsertSort.java

```
public class TestInsertSort {
    public static void main(String[] args) {
        int [] scores = { 80, 70, 60, 50, 95};
        sort (scores);
        for (int score : scores) {
            System.out .print(score + "," );
        }
    }
    public static void sort(int [] arrays) {
        for (int i = 1; i < arrays.length ; i++) {
            int insertElement = arrays[i];//Take unsorted new
elements
            int insertPosition = i;
            for (int j = insertPosition - 1; j >= 0; j--) {
                if (insertElement < arrays[j]) {// insertElement is
shifted to the right
                    arrays[j + 1] = arrays[j];
                    insertPosition--;
                }else {
                    break ;
                }
            }
            arrays[insertPosition] = insertElement;//Insert the new
element
        }
    }
}
```

Result:

50,60,70,80,95,

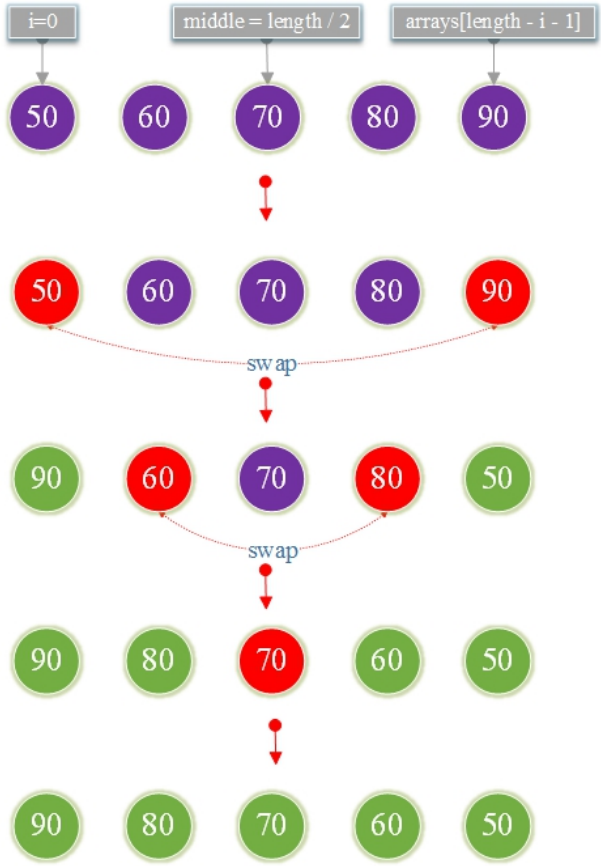
Reverse Array

Inversion of ordered sequences:

1 . Algorithmic ideas

Initial i = 0 and then swap the first element arrays[i] with last element arrays[length - i - 1]

Repeat until index of middle i == length / 2 .


i=0
middle = length / 2
arrays[length - i - 1]
50
60
70
80
90
50
60
70
80
90
swap
90
60
70
80
50
swap
90
80
70
60
50
90
80
70
60
50

TestReverse.java

```
public class TestReverse {
    public static void main(String[] args) {
        int [] scores = { 50, 60, 70, 80, 90 };
        reverse (scores);
        for (int score : scores) {
            System.out .print(score + "," );
        }
    }
    public static void reverse(int [] arrays) {
        int length = arrays.length ;
        int middle = length / 2;
        for (int i = 0; i <= middle; i++) {
            int temp = arrays[i];
            arrays[i] = arrays[length - i - 1];
            arrays[length - i - 1] = temp;
        }
    }
}
```

Result:

90,80,70,60,50,

Linear Table Search

1 . Please enter the value you want to search like : 70 return index.

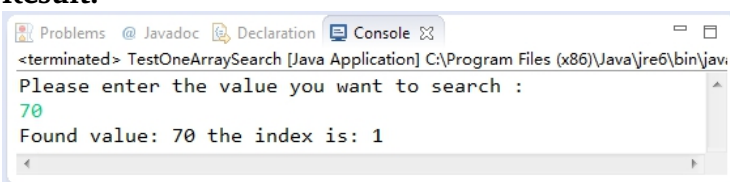
Analysis:

Traverse the value in the array scores, if there is a value equal to the given value like 70 , print out the current index

TestOneArraySearch.java

```
import java.util.Scanner;
public class TestOneArraySearch {
    public static int search(int [] array, int value){
        for (int i = 0; i < array.length ; i++) {
            if (array[i] == value) {
                return i;
            }
        }
        return -1;
    }
    public static void main(String[] args) {
        int [] scores = { 90, 70, 50, 80, 60, 85 };
        System.out .println("Please enter the value you want to
search :" );
        Scanner in = new Scanner(System.in );
        int value = in.nextInt();
        int index = search (scores, value);
        if (index > 0) {
            System.out .println("Found value: " + value + " the
index is: " + index);
        }else {
            System.out .println("The value was not found : " +
value);
        }
    }
}
```

Result:

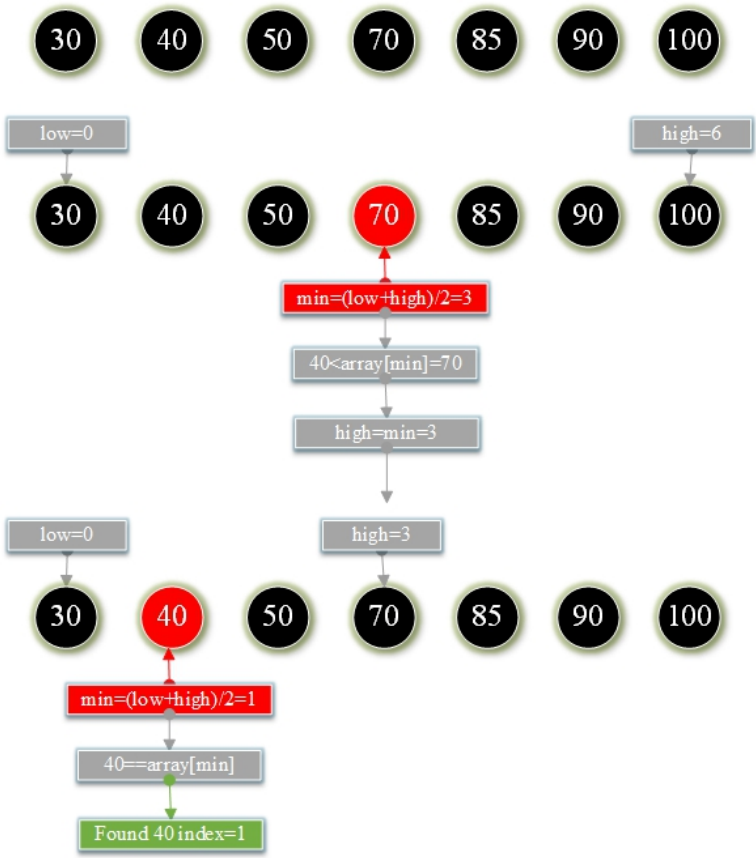


Dichotomy Binary Search

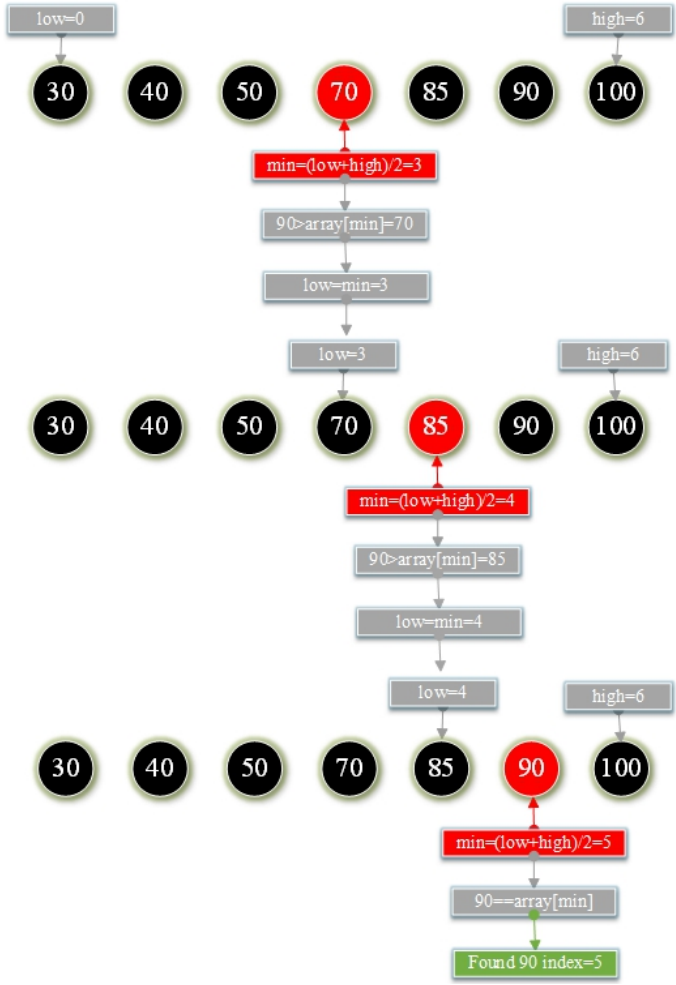
Dichotomy Binary Search:
Find the index position of a given value from an already ordered array.

1. Initialize the lowest index low = 0 , the highest index high = scores.length-1
2. Find the searchValue of the middle index mid = (low + high)/2 scores[mid]
3. Compare the scores[mid] with searchValue
   If the scores[mid] = = searchValue print current mid index,
   If scores[mid] > searchValue that the searchValue will be found between low and mid-1
4. And so on. Repeat step 3 until you find searchValue or low > = high to terminate the loop.

Example 1 : Find the index of searchValue = 40 in the array that has been sorted below.



Example 2 : Find the index of searchValue = 90 in the array that has been sorted below.



TestBinarySearch.java

```
public class TestBinarySearch {
    public static void main(String[] args) {
        int [] scores = { 30, 40, 50, 70, 85, 90, 100 };
        int searchValue = 40;
        int position = binarySearch (scores, searchValue);
        System.out .println(searchValue + " position:" + position);
        System.out .println("----------------------------" );
        searchValue = 90;
        position = binarySearch (scores, searchValue);
        System.out .println(searchValue + " position:" + position);
    }
    public static int binarySearch(int [] arrays, int searchValue)
{
        int low = 0;
        int high = arrays.length - 1;
        int mid = 0;
        while (low <= high) {
            mid = (low + high) / 2;
            if (arrays[mid] == searchValue) {
                return mid;
            } else if (arrays[mid] < searchValue) {
                low = mid + 1;
            } else if (arrays[mid] > searchValue) {
                high = mid - 1;
            }
        }
        return -1;
    }
}
```

Result:

```
40 position:1
----------------------------
90 position:5
```

Shell Sorting

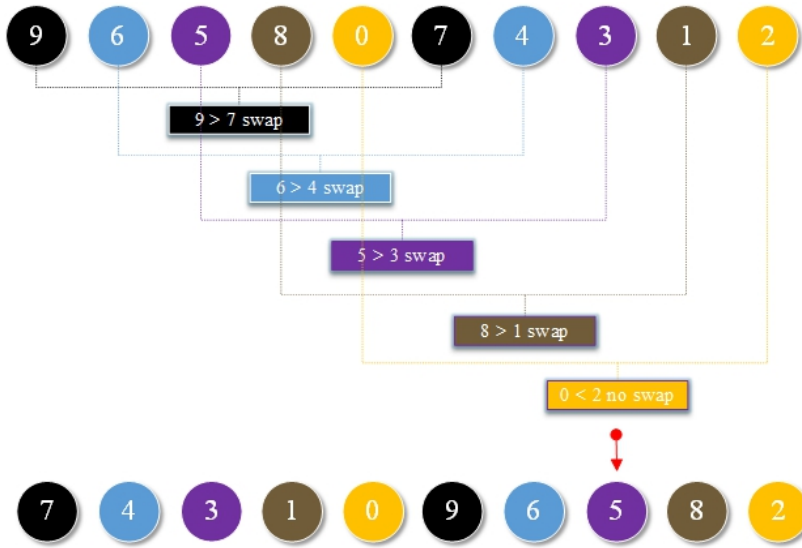
Shell Sorting:
Shell sort is a highly efficient sorting algorithm and is based on insertion sort algorithm. This algorithm avoids large shifts as in case of insertion sort, if the smaller value is to the far right and has to be moved to the far left.

Sort the following numbers from small to large by Shell Sorting

Algorithmic result:
The array is grouped according to a certain increment of subscripts, and the insertion of each group is sorted. As the increment decreases gradually until the increment is 1, the whole data is grouped and sorted.

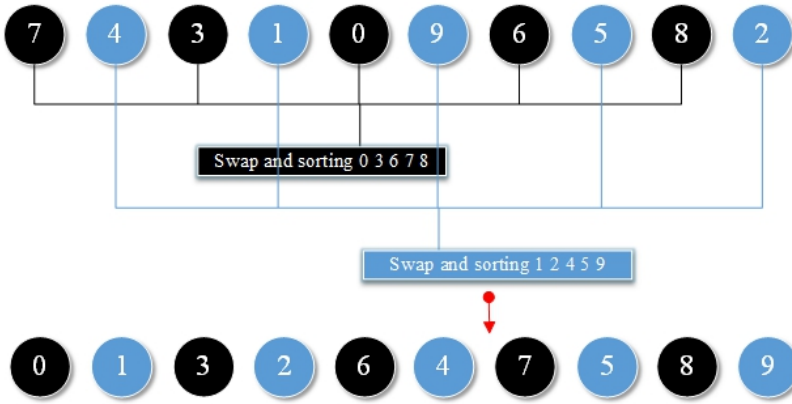
1 . The first sorting :

gap = array.length / 2 = 5



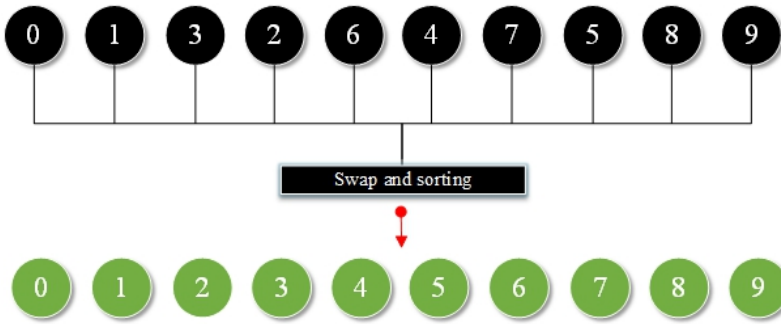
2 . The second sorting :

gap = 5 / 2 = 2



3 . The third sorting :

gap = 2 / 2 = 1



TestShellSort.java

```
public class TestShellSort {
    public static void main(String[] args) {
        int [] scores = { 9, 6, 5, 8, 0, 7, 4, 3, 1, 2 };
        shellSort (scores);
        for (int i = 0; i < scores.length ; i++) {
            System.out .print(scores[i] + "," );
        }
    }
    public static void shellSort(int [] array) {
        for (int gap = array.length / 2; gap > 0; gap = gap / 2) {
            for (int i = gap; i < array.length ; i++) {
                int j = i;
                while (j - gap >= 0 && array[j] < array[j - gap]) {
                    swap (array, j, j - gap);
                    j = j - gap;
                }
            }
        }
    }
    public static void swap(int [] array, int a, int b) {
        array[a] = array[a] + array[b];
        array[b] = array[a] - array[b];
        array[a] = array[a] - array[b];
    }
}
```

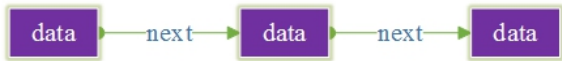
Result:

0,1,2,3,4,5,6,7,8,9,

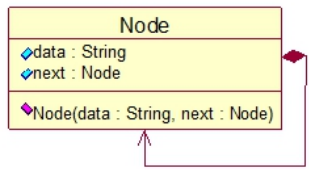
Single Link

Single Link:

Is a chained storage structure of a linear table, which is connected by a node. Each node consists of data and pointer to the next node.



UML Diagram

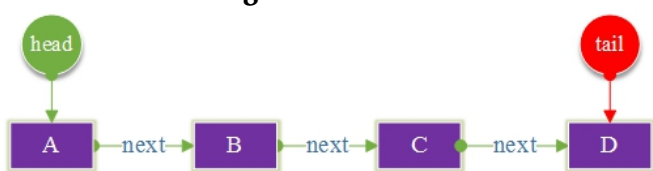


Node.java

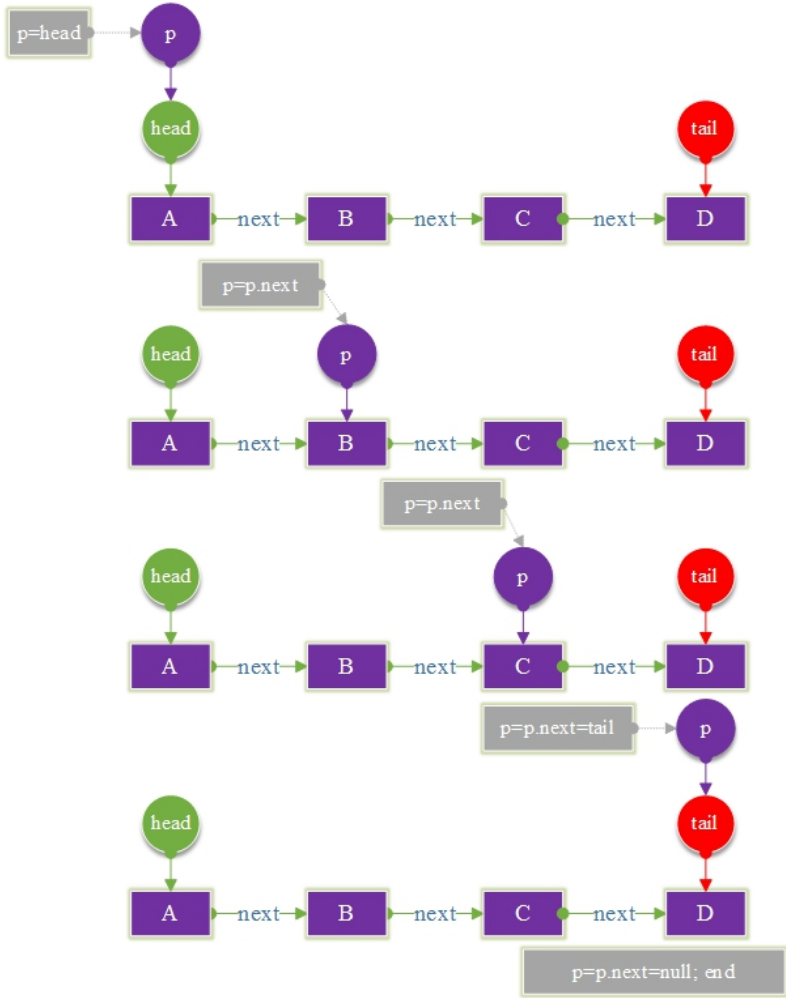
```
public class Node {
    public String data ;
    public Node next ;
    public Node(String data, Node next) {
            this .data = data;
            this .next = next;
    }
}
```

1. Single Link initialization .

Example : Construct a Single link list



2. traversal output .



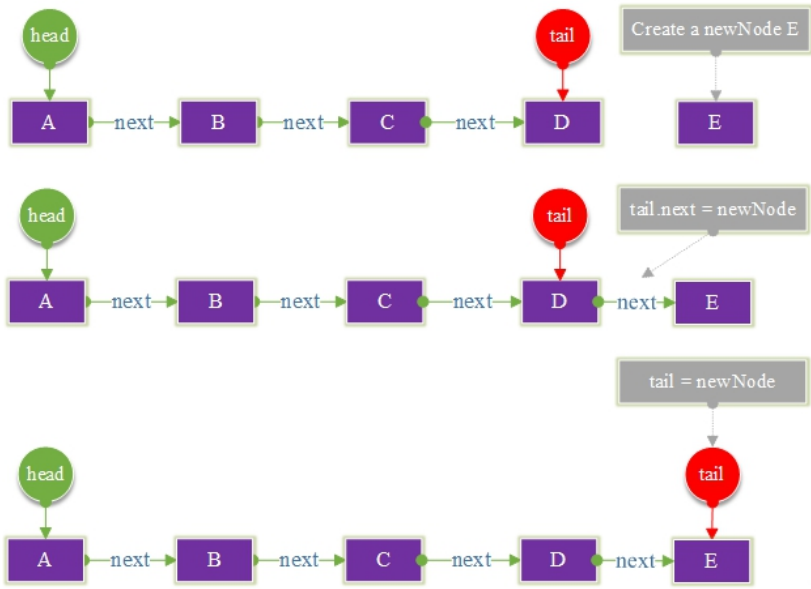
TestSingleLinkList.java

```
public class TestSingleLinkList {
    private static Node head ;
    private static Node tail ;
    public static Node init() {
        // the first node called head node
        head = new Node("A" , null );
        Node nodeB = new Node("B" , null );
        head .next = nodeB;
        Node nodeC = new Node("C" , null );
        nodeB.next = nodeC;
        // the last node called tail node
        tail = new Node("D" , null );
        nodeC.next = tail ;
        return head ;
    }
    public static void print() {
        Node p = head ;
        while (p != null ) // From the beginning to the end
        {
            System.out .print(p.data + " -> " );
            p = p.next ;
        }
        System.out .print("End\n\n" );
    }
    public static void main(String[] args) {
        init ();
        print ();
    }
}
```

Result:

A -> B -> C -> D -> End

3. Append a new node name: E to the end.



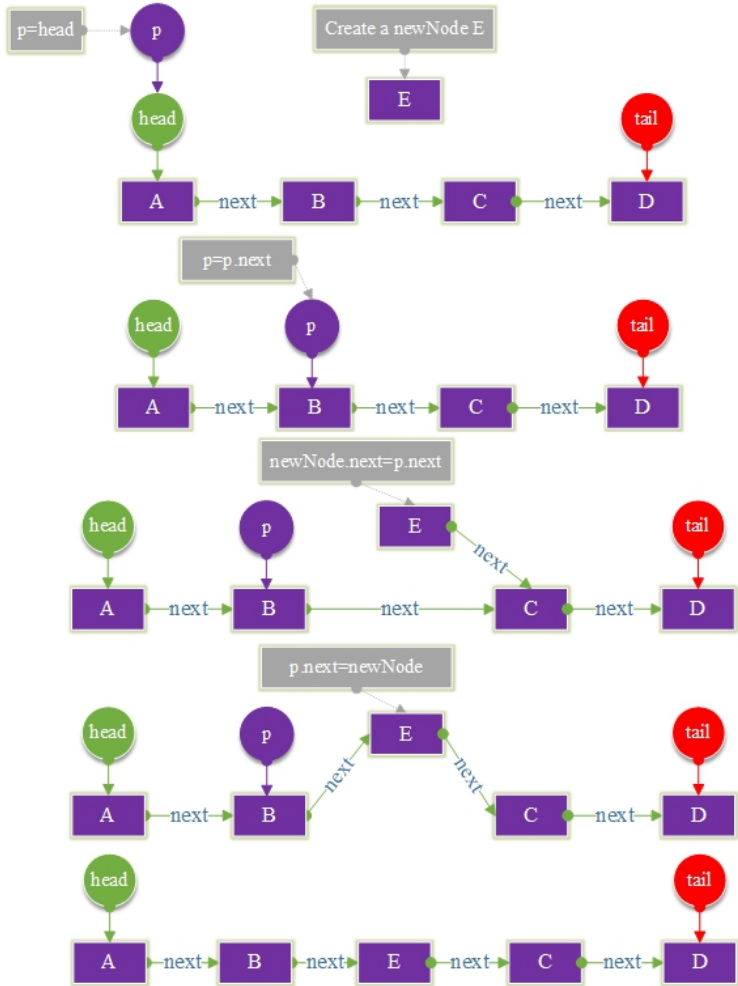
TestSingleLinkList.java

```
public class TestSingleLinkList {
    private static Node head ;
    private static Node tail ;
    public static Node init() {
        // the first node called head node
        head = new Node("A" , null );
        Node nodeB = new Node("B" , null );
        head .next = nodeB;
        Node nodeC = new Node("C" , null );
        nodeB.next = nodeC;
        // the last node called tail node
        tail = new Node("D" , null );
        nodeC.next = tail ;
        return head ;
    }
    public static void add(Node newNode) {
        tail .next = newNode;
        tail = newNode;
    }
    public static void print() {
        Node p = head ;
        while (p != null ) // From the beginning to the end
        {
            System.out .print(p.data + " -> " );
            p = p.next ;
        }
        System.out .print("End\n\n" );
    }
    public static void main(String[] args) {
        init ();
        System.out .println("Append a new node name: E  to the
end: " );
        add (new Node("E" , null ));
        print ();
    }
}
```

Result:

1. Append a new node name: E to the end:
2. A -> B -> C -> D -> E -> End

3. Insert a node Walnut in position 2.



TestSingleLinkList.java

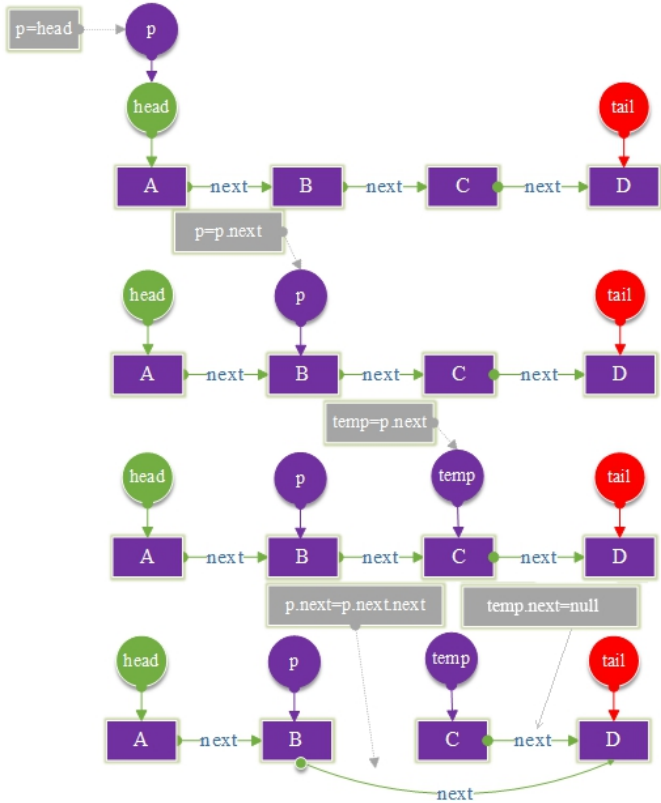
```
public class TestSingleLinkList {
    private static Node head ;
    private static Node tail ;
    public static Node init() {
        // the first node called head node
        head = new Node("A" , null );
        Node nodeB = new Node("B" , null );
        head .next = nodeB;
        Node nodeC = new Node("C" , null );
        nodeB.next = nodeC;
        // the last node called tail node
        tail = new Node("D" , null );
        nodeC.next = tail ;
        return head ;
    }
    public static void insert(int insertPosition, Node newNode) {
        Node p = head ;
        int i = 0;
        // Move the node to the insertion position
        while (p.next != null && i < insertPosition - 1) {
            p = p.next ;
            i++;
        }
        newNode.next = p.next ; // newNode next point to next
node
        p.next = newNode; // current next point to newNode
    }
    public static void print() {
        Node p = head ;
        while (p != null ) // From the beginning to the end
        {
            System.out .print(p.data + " -> " );
            p = p.next ;
        }
        System.out .print("End\n\n" );
    }
    public static void main(String[] args) {
        init ();
```

```
        System.out .println("Insert a new node E at index = 2 : " );
        insert (2, new Node("E" , null ));
        print ();
    }
}
```

Result:

```
Insert a new node E at index = 2 :
A -> B -> E -> C -> D -> End
```

4. Delete the index = 2 node.



TestSingleLinkList.java

```
public class TestSingleLinkList {
    private static Node head ;
    private static Node tail ;
    public static Node init() {
        // the first node called head node
        head = new Node("A" , null );
        Node nodeB = new Node("B" , null );
        head .next = nodeB;
        Node nodeC = new Node("C" , null );
        nodeB.next = nodeC;
        // the last node called tail node
        tail = new Node("D" , null );
        nodeC.next = tail ;
        return head ;
    }
    public static void remove(int removePosition) {
        Node p = head ;
        int i = 0;
        // Move the node to the previous node position that was
deleted
        while (p.next != null && i < removePosition - 1) {
            p = p.next ;
            i++;
        }
        Node temp = p.next ;// Save the node you want to delete
        p.next = p.next .next ;// Previous node next points to next
of delete the node
        temp.next = null ;
    }
    public static void print() {
        Node p = head ;
        while (p != null ) // From the beginning to the end
        {
            System.out .print(p.data + " -> " );
            p = p.next ;
        }
        System.out .print("End\n\n" );
    }
```

```
    public static void main(String[] args) {
        init ();
        System.out .println("Delete a new node C at index = 2 : " );
        remove (2);
        print ();
    }
}
```

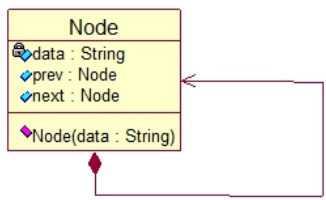
Result:

```
Delete a new node C at index = 2 :
A -> B -> D -> End
```

Doubly Linked List

Doubly Linked List:
It is a chained storage structure of a linear table. It is connected by nodes in two directions. Each node consists of data, pointing to the previous node and pointing to the next node.

UML Diagram

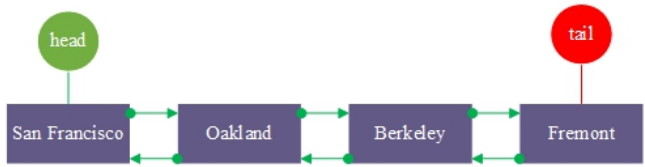


Node.java

```
class Node {
    private String data ;
    public Node prev ;
    public Node next ;
    public Node(String data) {
        this .data = data;
    }
    public String getData() {
        return data ;
    }
    public void setData(String data) {
        this .data = data;
    }
}
```

1. Doubly Linked List initialization .

Example : Construct a San Francisco subway Doubly linked list



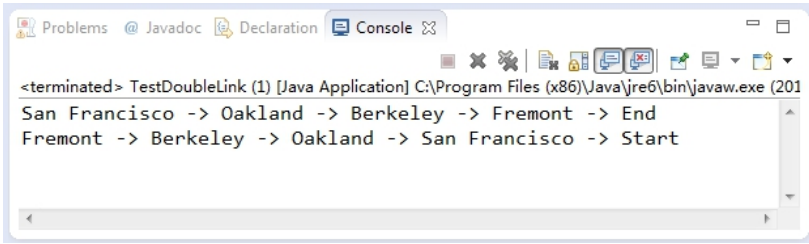
2. traversal output .

TestDoubleLink.java

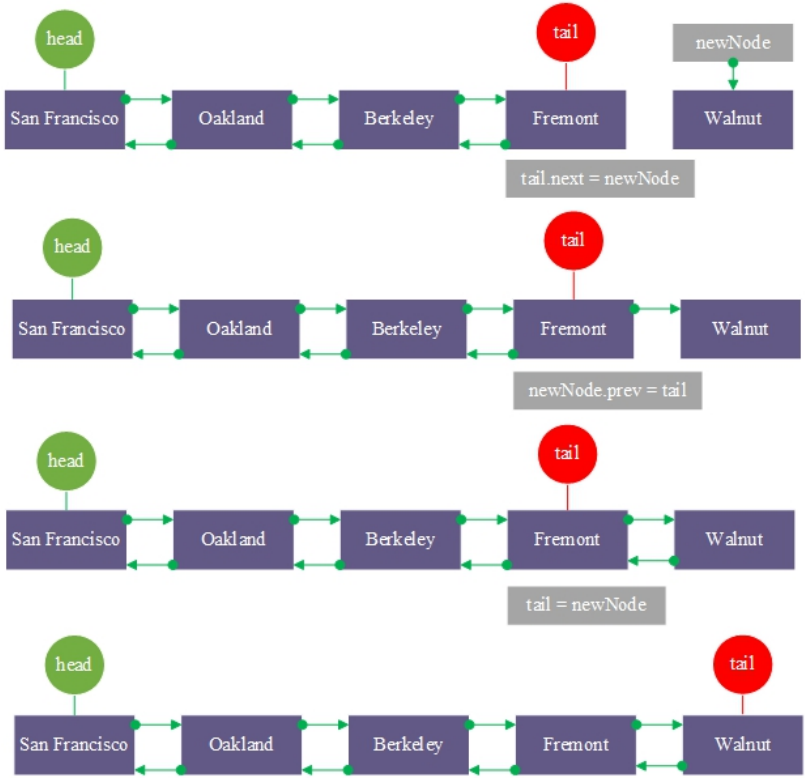
```
public class TestDoubleLink {
    private static Node head ;
    private static Node tail ;
    public static void init() {
        head = new Node("San Francisco" );
        head .prev = null ;
        head .next = null ;
        Node nodeOakland = new Node("Oakland" );
        nodeOakland.prev = head ;
        nodeOakland.next = null ;
        head .next = nodeOakland;
        Node nodeBerkeley = new Node("Berkeley" );
        nodeBerkeley.prev = nodeOakland;
        nodeBerkeley.next = null ;
        nodeOakland.next = nodeBerkeley;
        tail = new Node("Fremont" );
        tail .prev = nodeBerkeley;
        tail .next = null ;
        nodeBerkeley.next = tail ;
    }
    public static void print(Node node) {
        Node p = node;
        Node end = null ;
        while (p != null )
        {
            String data = p.getData();
            System.out .print(data + " -> " );
```

```
            end = p;
            p = p.next ;
        }
        System.out .print("End\n" );
        p = end;
        while (p != null )
        {
            String data = p.getData();
            System.out .print(data + " -> " );
            p = p.prev ;
        }
        System.out .print("Start\n\n" );
    }
    public static void main(String[] args) {
        init ();
        print (head );
    }
}
```

Result:


Problems
Javadoc
Declaration
Console
<terminated> TestDoubleLink (1) [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (201
San Francisco -> Oakland -> Berkeley -> Fremont -> End
Fremont -> Berkeley -> Oakland -> San Francisco -> Start

3. add a node Walnut at the end of Fremont.



TestDoubleLink.java

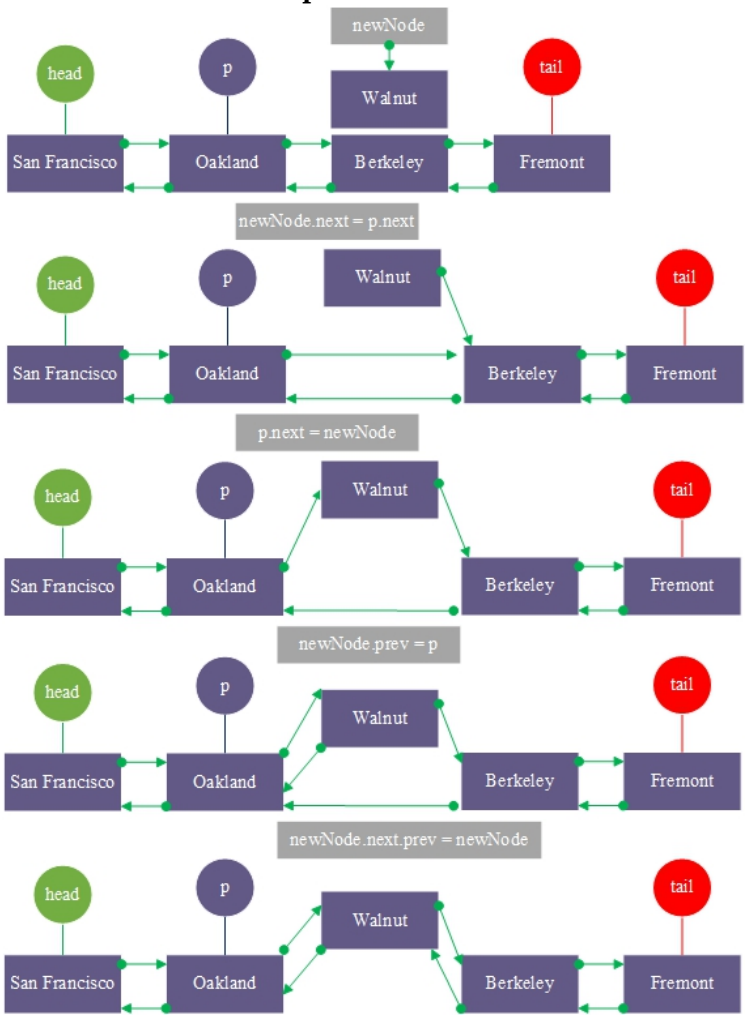
```
public class TestDoubleLink {
    private static Node head ;
    private static Node tail ;
    public static void init() {
        head = new Node("San Francisco" );
        head .prev = null ;
        head .next = null ;
        Node nodeOakland = new Node("Oakland" );
        nodeOakland.prev = head ;
        nodeOakland.next = null ;
        head .next = nodeOakland;
        Node nodeBerkeley = new Node("Berkeley" );
        nodeBerkeley.prev = nodeOakland;
        nodeBerkeley.next = null ;
        nodeOakland.next = nodeBerkeley;
        tail = new Node("Fremont" );
        tail .prev = nodeBerkeley;
        tail .next = null ;
        nodeBerkeley.next = tail ;
    }
    public static void add(Node newNode) {
        tail .next = newNode;
        newNode.prev = tail ;
        tail = newNode;
    }
    public static void print(Node node) {
        Node p = node;
        Node end = null ;
        while (p != null ) {
            String data = p.getData();
            System.out .print(data + " -> " );
            end = p;
            p = p.next ;
        }
        System.out .print("End\n" );
        p = end;
        while (p != null ) {
            String data = p.getData();
```

```
            System.out .print(data + " -> " );
            p = p.prev ;
        }
        System.out .print("Start\n\n" );
    }
    public static void main(String[] args) {
        init ();
        add (new Node("Walnut" ));
        print (head );
    }
}
```

Result:

Problems
Javadoc
Declaration
Console
<terminated> TestDoubleLink (1) [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (2019年
San Francisco -> Oakland -> Berkeley -> Fremont -> Walnut -> End
Walnut -> Fremont -> Berkeley -> Oakland -> San Francisco -> Start

3. Insert a node Walnut in position 2.



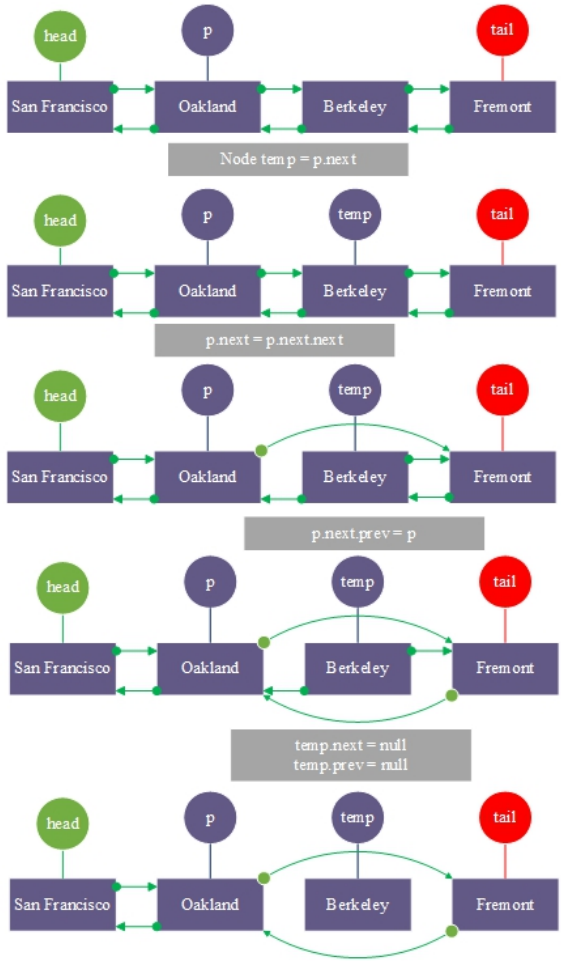
TestDoubleLink.java

```
public class TestDoubleLink {
    private static Node head ;
    private static Node tail ;
    public static void init() {
        head = new Node("San Francisco" );
        head .prev = null ;
        head .next = null ;
        Node nodeOakland = new Node("Oakland" );
        nodeOakland.prev = head ;
        nodeOakland.next = null ;
        head .next = nodeOakland;
        Node nodeBerkeley = new Node("Berkeley" );
        nodeBerkeley.prev = nodeOakland;
        nodeBerkeley.next = null ;
        nodeOakland.next = nodeBerkeley;
        tail = new Node("Fremont" );
        tail .prev = nodeBerkeley;
        tail .next = null ;
        nodeBerkeley.next = tail ;
    }
    public static void insert(int insertPosition, Node newNode) {
        Node p = head ;
        int i = 0;
        // Move the node to the insertion position
        while (p.next != null && i < insertPosition-1) {
            p = p.next ;
            i++;
        }
        newNode.next = p.next ;
        p.next = newNode;
        newNode.prev = p;
        newNode.next .prev = newNode;
    }
    public static void print(Node node) {
        Node p = node;
        Node end = null ;
        while (p != null ) {
            String data = p.getData();
```

```
            System.out .print(data + " -> " );
            end = p;
            p = p.next ;
        }
        System.out .print("End\n" );
        p = end;
        while (p != null ) {
            String data = p.getData();
            System.out .print(data + " -> " );
            p = p.prev ;
        }
        System.out .print("Start\n\n" );
    }
    public static void main(String[] args) {
        init ();
        insert (2,new Node("Walnut" ));
        print (head );
    }
}
```

Result:

4. Delete the index = 2 node.

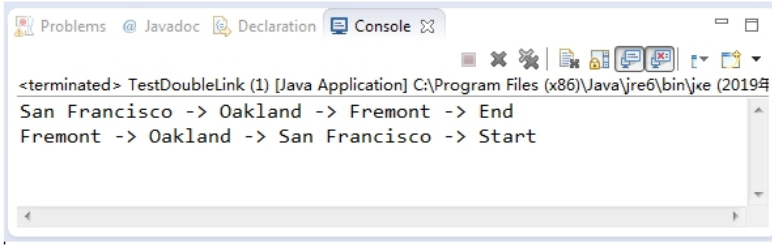


TestDoubleLink.java

```
public class TestDoubleLink {
    private static Node head ;
    private static Node tail ;
    public static void init() {
        head = new Node("San Francisco" );
        head .prev = null ;
        head .next = null ;
        Node nodeOakland = new Node("Oakland" );
        nodeOakland.prev = head ;
        nodeOakland.next = null ;
        head .next = nodeOakland;
        Node nodeBerkeley = new Node("Berkeley" );
        nodeBerkeley.prev = nodeOakland;
        nodeBerkeley.next = null ;
        nodeOakland.next = nodeBerkeley;
        tail = new Node("Fremont" );
        tail .prev = nodeBerkeley;
        tail .next = null ;
        nodeBerkeley.next = tail ;
    }
    public static void remove(int removePosition) {
        Node p = head ;
        int i = 0;
        // Move the node to the previous node  that was deleted
        while (p.next != null && i < removePosition - 1) {
            p = p.next ;
            i++;
        }
        Node temp = p.next ;// Save the node you want to delete
        p.next = p.next .next ;
        p.next .prev = p;
        temp.next = null ; // Set the delete node next to null
        temp.prev = null ;// Set the delete node prev to null
    }
    public static void print(Node node) {
        Node p = node;
        Node end = null ;
        while (p != null ) {
```

```
            String data = p.getData();
            System.out.print(data + " -> " );
            end = p;
            p = p.next ;
        }
        System.out.print("End\n" );
        p = end;
        while (p != null ) {
            String data = p.getData();
            System.out.print(data + " -> " );
            p = p.prev ;
        }
        System.out.print("Start\n\n" );
    }
    public static void main(String[] args) {
        init ();
        remove (2);
        print (head );
    }
}
```

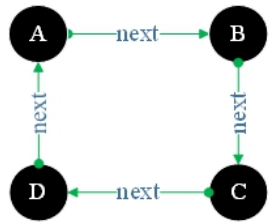
Result:


Problems
Javadoc
Declaration
Console
<terminated> TestDoubleLink (1) [Java Application] C:\Program Files (x86)\Java\jre6\bin\jxe (2019年
San Francisco -> Oakland -> Fremont -> End
Fremont -> Oakland -> San Francisco -> Start

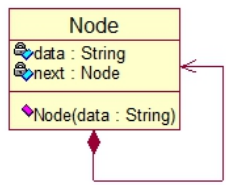
One-way Circular LinkedList

One-way Circular List:

It is a chain storage structure of a linear table, which is connected to form a ring, and each node is composed of data and a pointer to next.



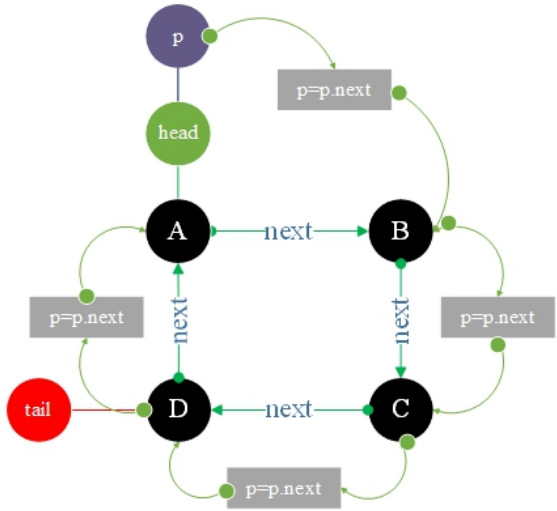
UML Diagram



Node.java

```
public class Node {
    private String data ;
    public Node next ;
    public Node(String data) {
        this .data = data;
    }
    public String getData() {
        return data ;
    }
    public void setData(String data) {
        this .data = data;
    }
}
```

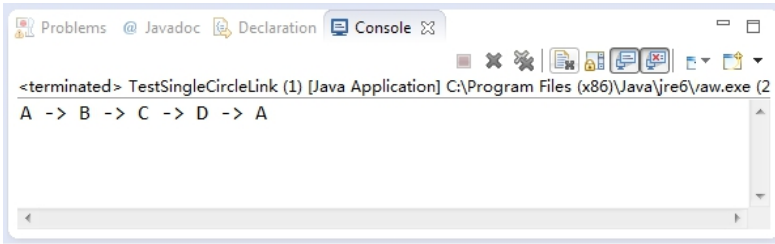
1. One-way Circular Linked List initialization and traversal output .



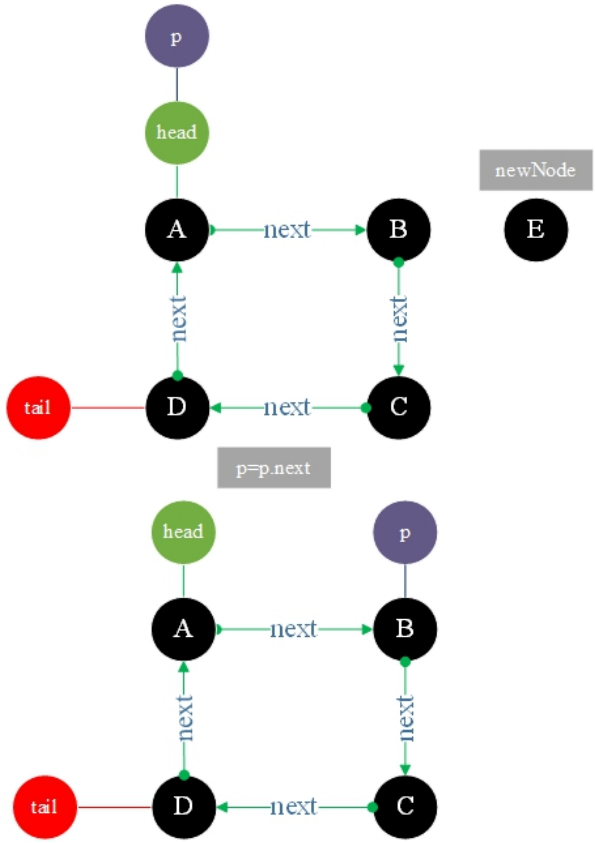
TestSingleCircleLink.java

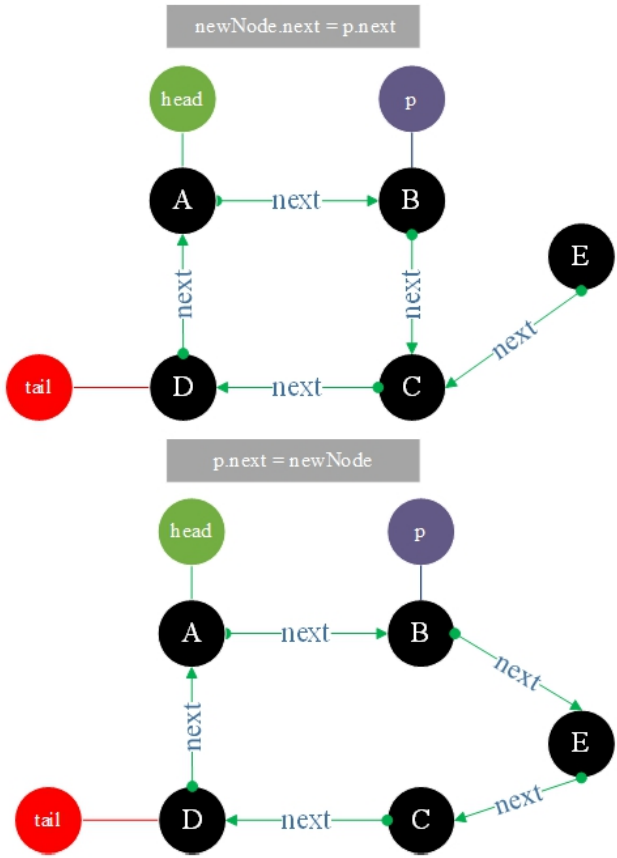
```
public class TestSingleCircleLink {
    private static Node head ;
    private static Node tail ;
    public static void init() {
        // the first node called head node
        head = new Node("A" );
        head .next = null ;
        Node nodeB = new Node("B" );
        nodeB.next = null ;
        head .next = nodeB;
        Node nodeC = new Node("C" );
        nodeC.next = null ;
        nodeB.next = nodeC;
        // the last node called tail node
        tail = new Node("D" );
        tail .next = head ;
        nodeC.next = tail ;
    }
    public static void print() {
        Node p = head ;
        do {
            String data = p.getData();
            System.out .print(data + " -> " );
            p = p.next ;
        }while (p!=head );
        String data = p.getData();
        System.out .print(data + "\n\n" );
    }
    public static void main(String[] args) {
        init ();
        print ();
    }
}
```

Result:


Problems
Javadoc
Declaration
Console
<terminated> TestSingleCircleLink (1) [Java Application] C:\Program Files (x86)\Java\jre6\jaw.exe (2
A -> B -> C -> D -> A

3. Insert a node E in position 2.




newNode.next = p.next
head
p
A
next
B
E
next
next
next
tail
D
next
C
p.next = newNode
head
p
A
next
B
next
E
next
next
tail
D
next
C

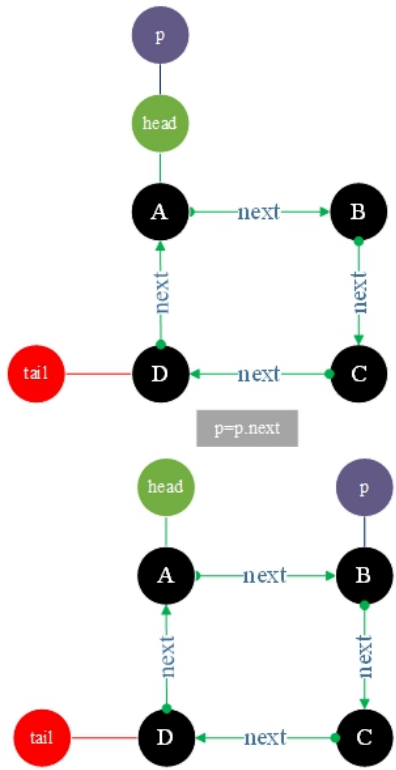
TestSingleCircleLink.java

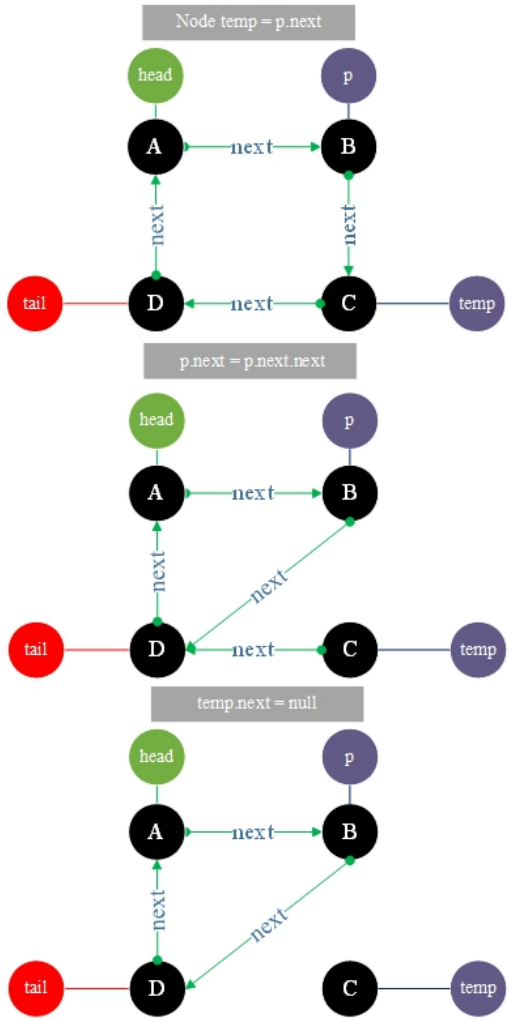
```
public class TestSingleCircleLink {
    private static Node head ;
    private static Node tail ;
    public static void init() {
        // the first node called head node
        head = new Node("A" );
        head .next = null ;
        Node nodeB = new Node("B" );
        nodeB.next = null ;
        head .next = nodeB;
        Node nodeC = new Node("C" );
        nodeC.next = null ;
        nodeB.next = nodeC;
        // the last node called tail node
        tail = new Node("D" );
        tail .next = head ;
        nodeC.next = tail ;
    }
    public static void insert(int insertPosition, Node newNode) {
        Node p = head ;
        int i = 0;
        // Move the node to the insertion position
        while (p.next != null && i < insertPosition - 1) {
            p = p.next ;
            i++;
        }
        newNode.next = p.next ;
        p.next = newNode;
    }
    public static void print() {
        Node p = head ;
        do {
            String data = p.getData();
            System.out .print(data + " -> " );
            p = p.next ;
        } while (p != head );
        String data = p.getData();
        System.out .print(data + "\n\n" );
```

```
    }
    public static void main(String[] args) {
        init ();
        insert (2,new Node("E" ));
        print ();
    }
}
```

Result:

4. Delete the index = 2 node.



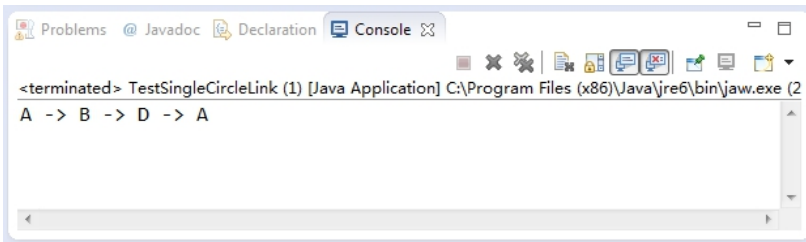

Node temp = p.next
head
p
A
next
B
next
next
tail
D
next
C
temp
p.next = p.next.next
head
p
A
next
B
next
next
tail
D
next
C
temp
temp.next = null
head
p
A
next
B
next
next
tail
D
C
temp

TestSingleCircleLink.java

```
public class TestSingleCircleLink {
    private static Node head ;
    private static Node tail ;
    public static void init() {
        // the first node called head node
        head = new Node("A" );
        head .next = null ;
        Node nodeB = new Node("B" );
        nodeB.next = null ;
        head .next = nodeB;
        Node nodeC = new Node("C" );
        nodeC.next = null ;
        nodeB.next = nodeC;
        // the last node called tail node
        tail = new Node("D" );
        tail .next = head ;
        nodeC.next = tail ;
    }
    public static void remove(int removePosition) {
        Node p = head ;
        int i = 0;
        while (p.next != null && i < removePosition - 1) {
            p = p.next ;
            i++;
        }
        Node temp = p.next ;
        p.next = p.next .next ;
        temp.next = null ;
    }
    public static void print() {
        Node p = head ;
        do {
            String data = p.getData();
            System.out .print(data + " -> " );
            p = p.next ;
        } while (p != head );
        String data = p.getData();
        System.out .print(data + "\n\n" );
```

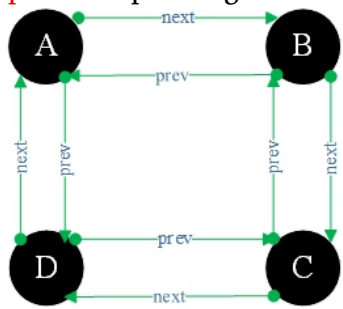
```
    }
    public static void main(String[] args) {
        init ();
        remove (2);
        print ();
    }
}
```

Result:

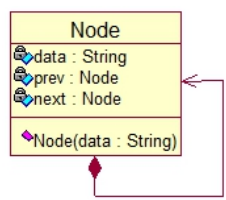


Two-way Circular LinkedList

Two-way Circular List: It is a chain storage structure of a linear table. The nodes are connected in series by two directions, and is connected to form a ring. Each node is composed of data , pointing to the previous node prev and pointing to the next node next .



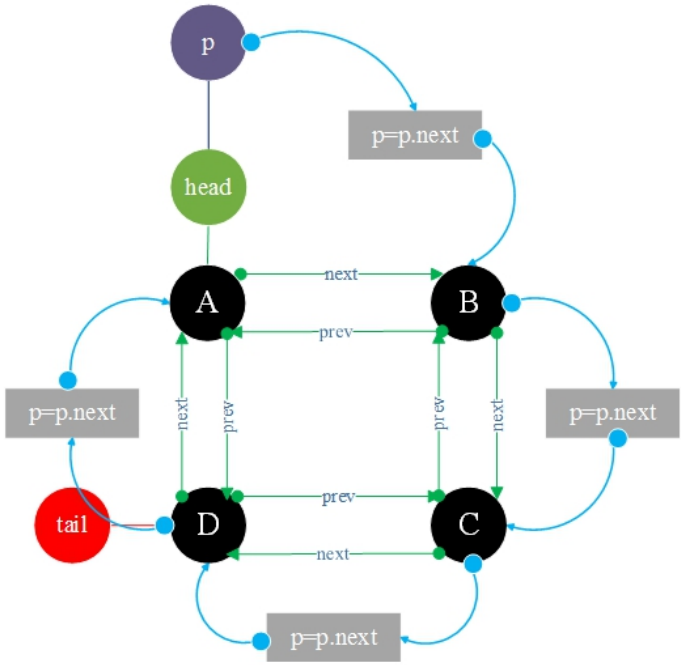
UML Diagram

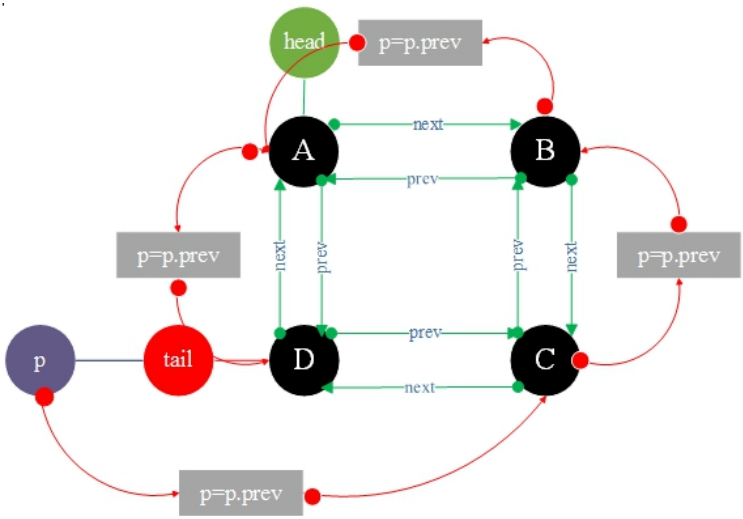


Node.java

```
class Node {
    private String data ;
    public Node prev ;
    public Node next ;
    public Node(String data) {
        this .data = data;
    }
    public String getData() {
        return data ;
    }
    public void setData(String data) {
        this .data = data;
    }
}
```

1. Two-way Circular Linked List initialization and traversal output .



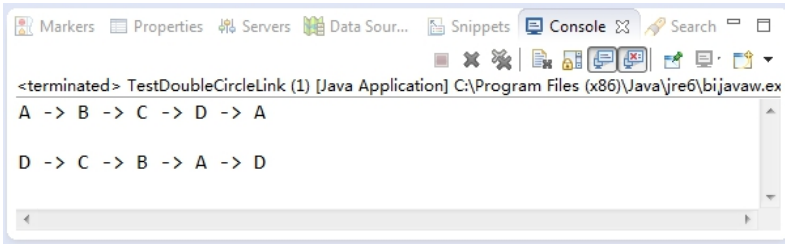

head
p=p.prev
A
next
B
prev
p=p.prev
next
prev
prev
next
p=p.prev
p
tail
D
prev
C
next
p=p.prev

TestDoubleCircleLink.java

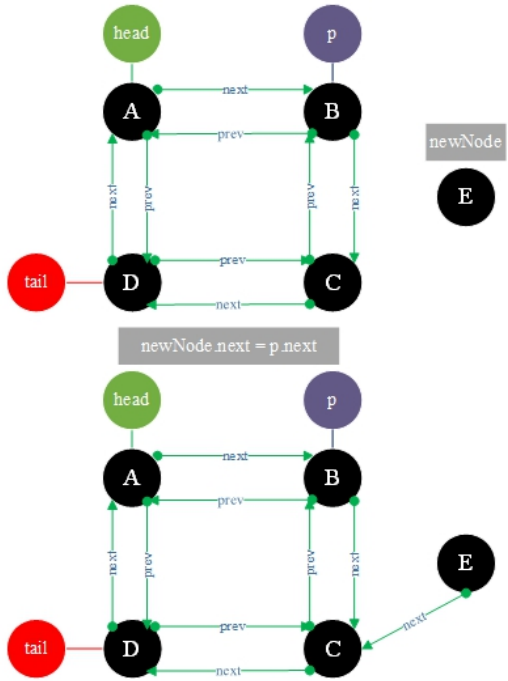
```
public class TestDoubleCircleLink {
    private static Node head ;
    private static Node tail ;
    public static void init() {
        // the first node called head node
        head = new Node("A" );
        head .next = null ;
        head .prev = null ;
        Node nodeB = new Node("B" );
        nodeB.next = null ;
        nodeB.prev = head ;
        head .next = nodeB;
        Node nodeC = new Node("C" );
        nodeC.next = null ;
        nodeC.prev = nodeB;
        nodeB.next = nodeC;
        // the last node called tail node
        tail = new Node("D" );
        tail .next = head ;
        tail .prev = nodeC;
        nodeC.next = tail ;
        head .prev = tail ;
    }
    public static void print() {
        Node p = head ;
        do {
            String data = p.getData();
            System.out .print(data + " -> " );
            p = p.next ;
        } while (p != head );
        String data = p.getData();
        System.out .print(data + "\n\n" );
        p = tail ;
        do {
            data = p.getData();
            System.out .print(data + " -> " );
            p = p.prev ;
        } while (p != tail );
```

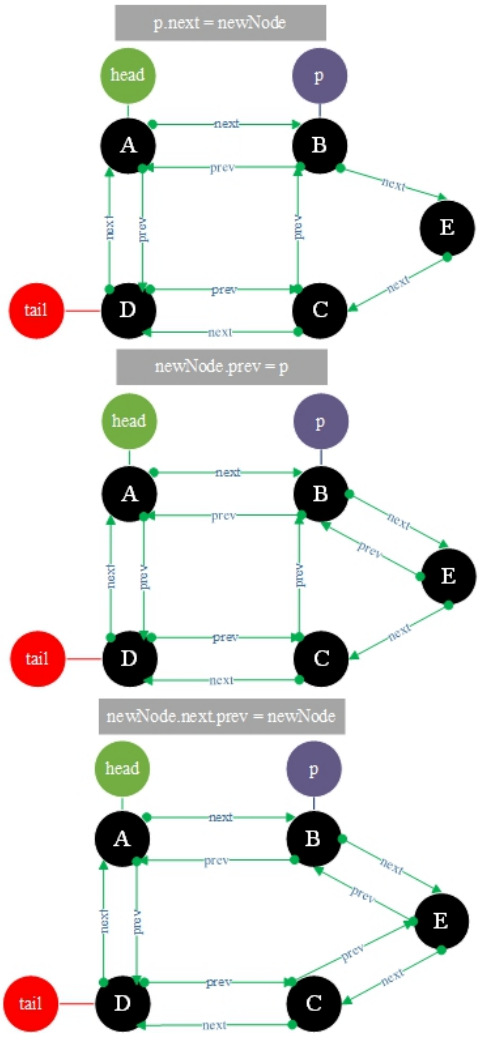
```
        data = p.getData();
        System.out.print(data + "\n\n");
    }
    public static void main(String[] args) {
        init();
        print();
    }
}
```

Result:



3. Insert a node E in position 2.



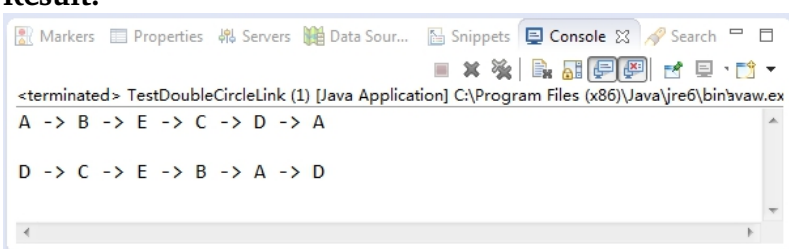

p.next = newNode
head
p
A
B
E
tail
D
C
next
prev
newNode.prev = p
newNode.next.prev = newNode

TestDoubleCircleLink.java

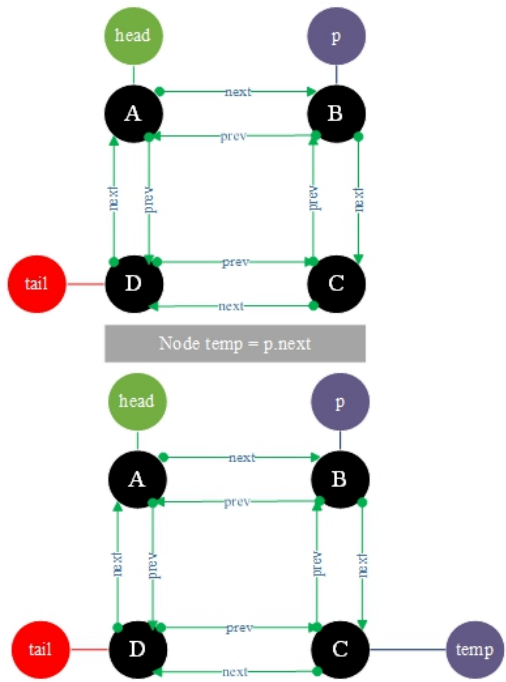
```
public class TestDoubleCircleLink {
    private static Node head ;
    private static Node tail ;
    public static void init() {
        // the first node called head node
        head = new Node("A" );
        head .next = null ;
        head .prev = null ;
        Node nodeB = new Node("B" );
        nodeB.next = null ;
        nodeB.prev = head ;
        head .next = nodeB;
        Node nodeC = new Node("C" );
        nodeC.next = null ;
        nodeC.prev = nodeB;
        nodeB.next = nodeC;
        // the last node called tail node
        tail = new Node("D" );
        tail .next = head ;
        tail .prev = nodeC;
        nodeC.next = tail ;
        head .prev = tail ;
    }
    public static void insert(int insertPosition, Node newNode) {
        Node p = head ;
        int i = 0;
        //Move the node to the insertion position
        while (p.next != null && i < insertPosition - 1) {
            p = p.next ;
            i++;
        }
        newNode.next = p.next ;
        p.next = newNode;
        newNode.prev = p;
        newNode.next .prev = newNode;
    }
    public static void print() {
        Node p = head ;
```

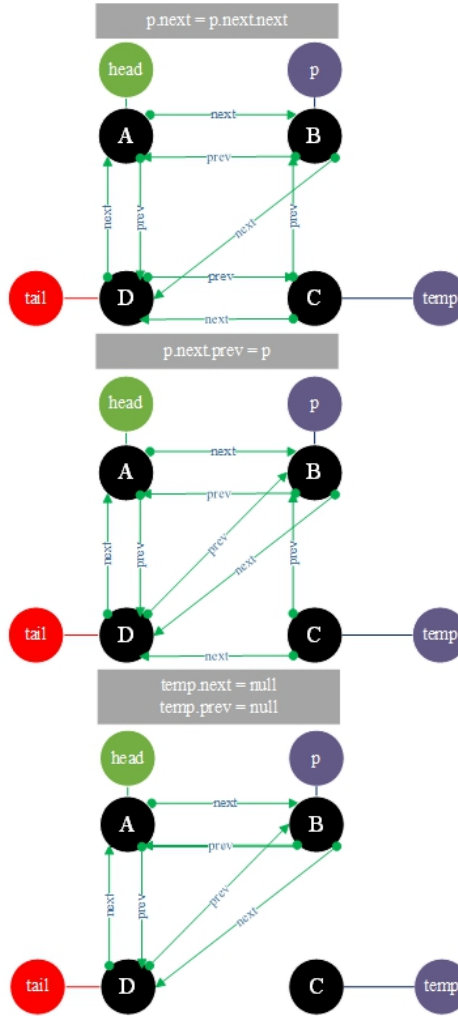
```
        do {
            String data = p.getData();
            System.out.print(data + " -> " );
            p = p.next ;
        } while (p != head );
        String data = p.getData();
        System.out.print(data + "\n\n" );
        p = tail ;
        do {
            data = p.getData();
            System.out.print(data + " -> " );
            p = p.prev ;
        } while (p != tail );
        data = p.getData();
        System.out.print(data + "\n\n" );
    }
    public static void main(String[] args) {
        init ();
        insert (2,new Node("E" ));
        print ();
    }
}
```

Result:



4. Delete the index = 2 node.



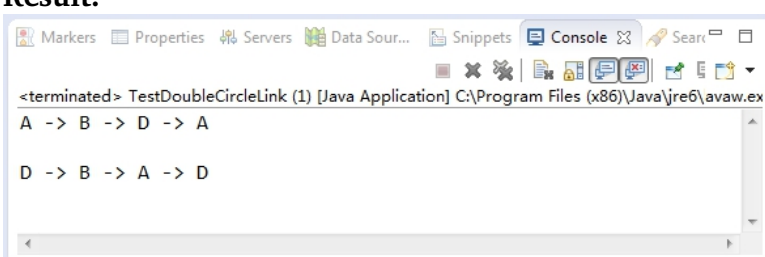

p.next = p.next.next
head
p
A
B
next
prev
next
prev
next
prev
tail
D
C
temp
next
p.next.prev = p
head
p
A
B
next
prev
prev
next
prev
next
tail
D
C
temp
next
temp.next = null
temp.prev = null
head
p
A
B
next
prev
prev
next
next
prev
tail
D
C
temp

TestDoubleCircleLink.java

```
public class TestDoubleCircleLink {
    private static Node head ;
    private static Node tail ;
    public static void init() {
        head = new Node("A" );
        head .next = null ;
        head .prev = null ;
        Node nodeB = new Node("B" );
        nodeB.next = null ;
        nodeB.prev = head ;
        head .next = nodeB;
        Node nodeC = new Node("C" );
        nodeC.next = null ;
        nodeC.prev = nodeB;
        nodeB.next = nodeC;
        // the last node called tail node
        tail = new Node("D" );
        tail .next = head ;
        tail .prev = nodeC;
        nodeC.next = tail ;
        head .prev = tail ;
    }
    public static void remove(int removePosition) {
        Node p = head ;
        int i = 0;
        while (p.next != null && i < removePosition - 1) {
            p = p.next ;
            i++;
        }
        Node temp = p.next ;
        p.next = p.next .next ;
        p.next .prev = p;
        temp.next = null ;//Set the delete node next to null
        temp.prev = null ;// Set the delete node prev to null
    }
    public static void print() {
        Node p = head ;
        do {
```

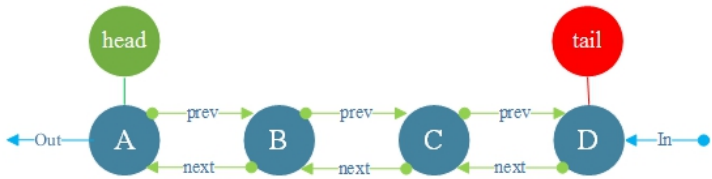
```
            String data = p.getData();
            System.out.print(data + " -> " );
            p = p.next ;
        } while (p != head );
        String data = p.getData();
        System.out.print(data + "\n\n" );
        p = tail ;
        do {
            data = p.getData();
            System.out.print(data + " -> " );
            p = p.prev ;
        } while (p != tail );
        data = p.getData();
        System.out.print(data + "\n\n" );
    }
    public static void main(String[] args) {
        init ();
        remove (2);
        print ();
    }
}
```

Result:

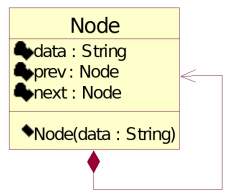


Queue

Queue: FIFO (First In First Out) sequence.



UML Diagram

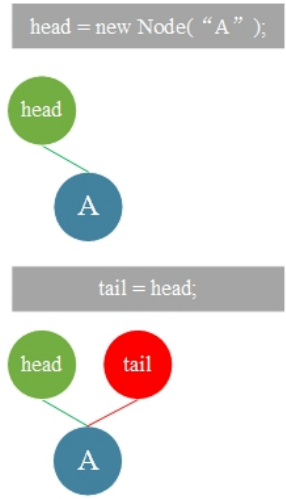


Node.java

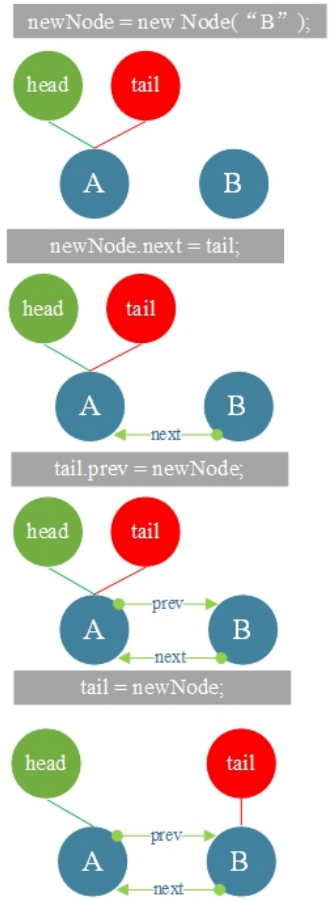
```
class Node {
    private String data ;
    public Node next ;
    public Node prev ;
    public Node(String data) {
        this .data = data;
    }
    public String getData() {
        return data ;
    }
    public void setData(String data) {
        this .data = data;
    }
}
```

1. Queue initialization and traversal output .

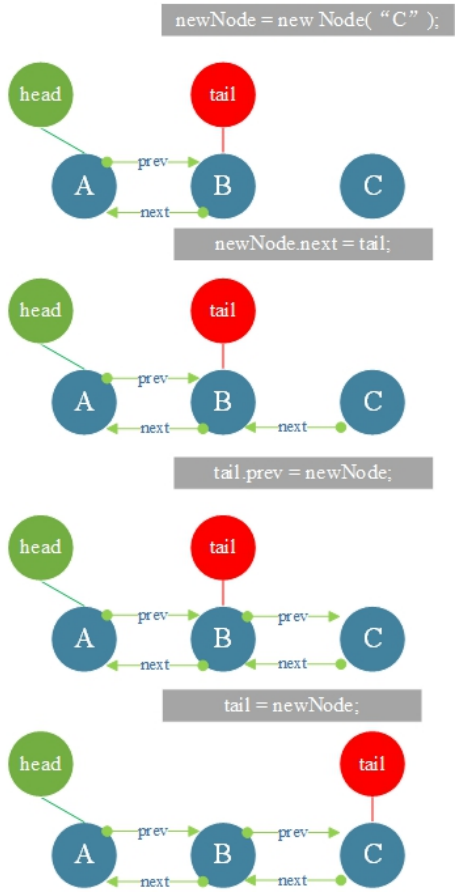
Initialization Insert A



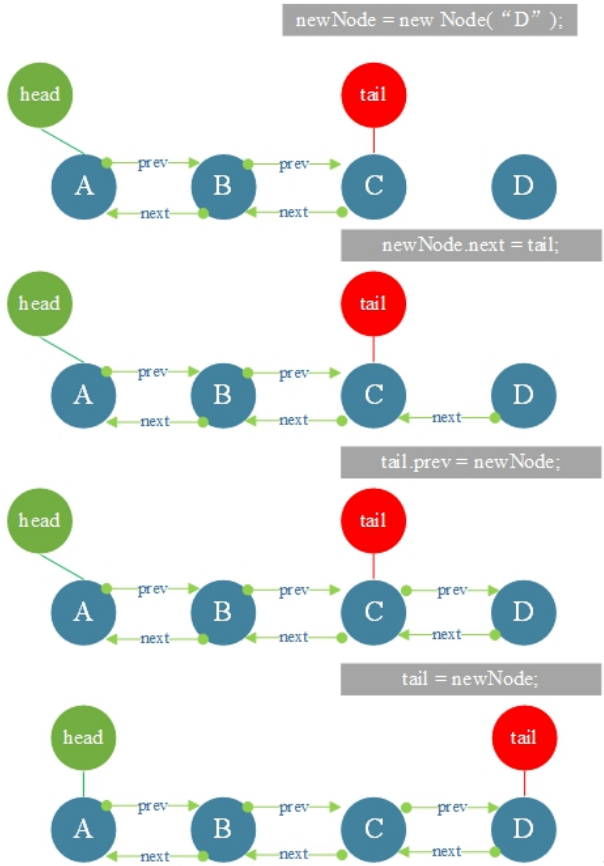
Initialization Insert B



Initialization Insert C



Initialization Insert D



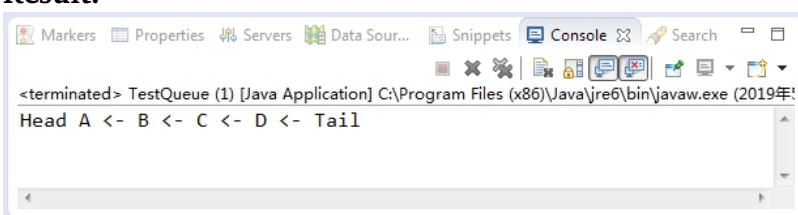
Queue.java

```
public class Queue {
    private Node head ;
    private Node tail ;
    private int size ;
    public void offer(String element) {
        if (head == null ) {
            head = new Node(element);
            tail = head ;
        } else {
            Node newNode = new Node(element);
            newNode.next = tail ;
            tail .prev = newNode;
            tail = newNode;
        }
        size ++;
    }
    public Node poll() {
        Node p = head ;
        if (p == null ) {
            return null ;
        }
        head = head .prev ;
        p.next = null ;
        p.prev = null ;
        size --;
        return p;
    }
    public int size() {
        return size ;
    }
}
```

TestQueue.java

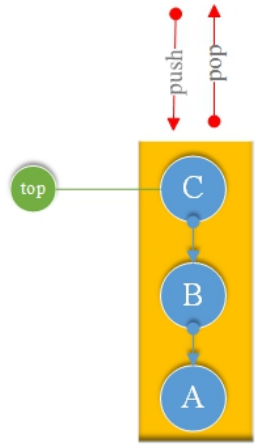
```
public class TestQueue {
    public static void main(String[] args) {
        Queue queue = new Queue();
        queue.offer("A" );
        queue.offer("B" );
        queue.offer("C" );
        queue.offer("D" );
        print (queue);
    }
    public static void print(Queue queue) {
        System.out .print("Head " );
        Node node = null ;
        while ((node = queue.poll())!=null ) {
            System.out .print(node.getData() + " <- " );
        }
        System.out .print("Tail\n" );
    }
}
```

Result:

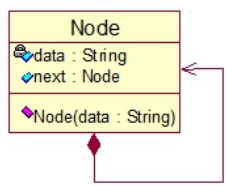


Stack

Stack: FILO (First In Last Out) sequence.



UML Diagram

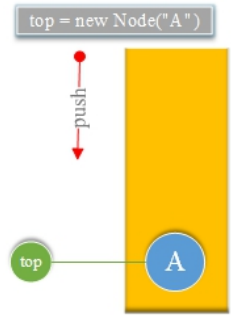


Node.java

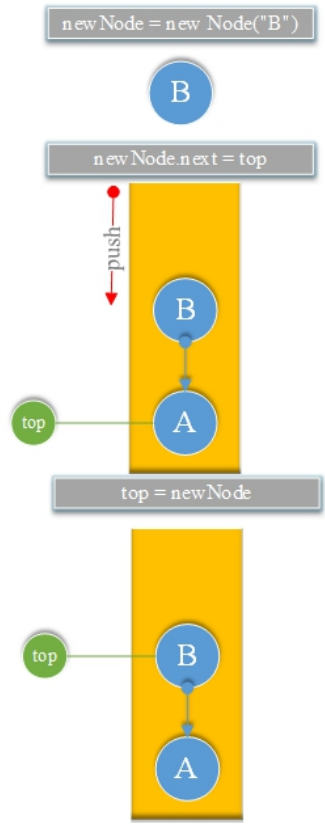
```
class Node {
    public String data ;
    public Node next ;
    public Node(String data) {
        this .data = data;
    }
}
```

1. Stack initialization and traversal output .

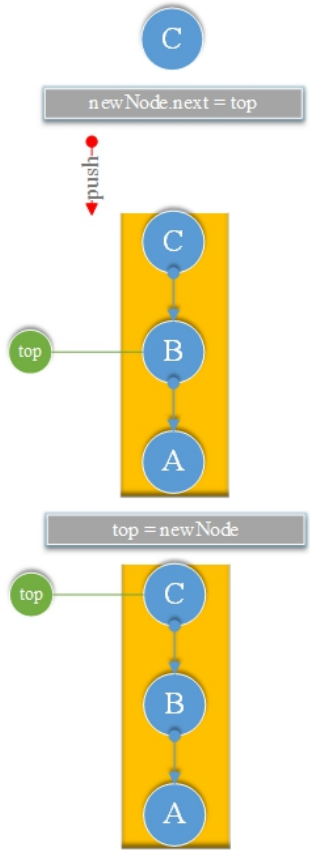
Push A into Stack



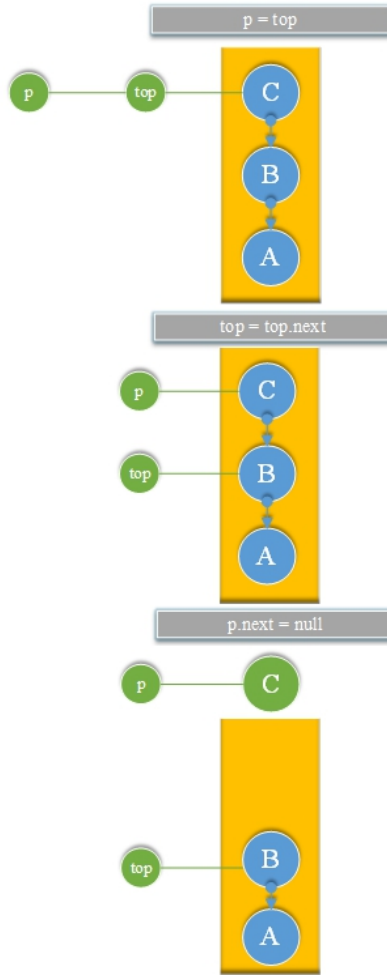
Push B into Stack



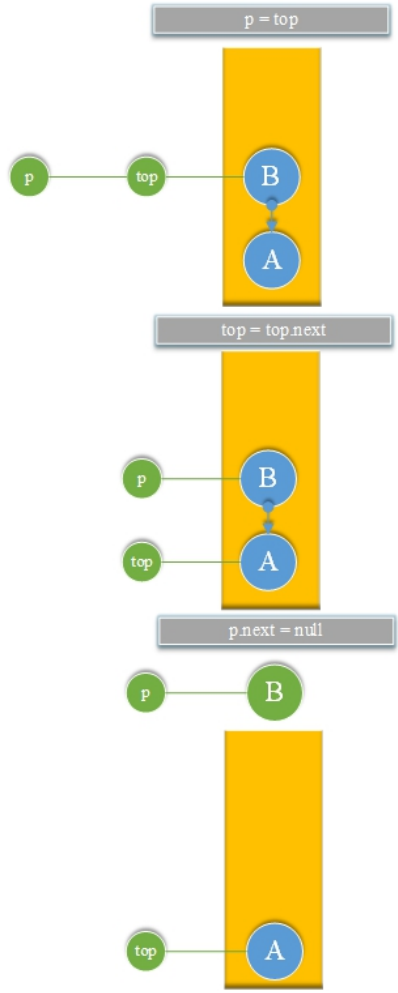
Push C into Stack



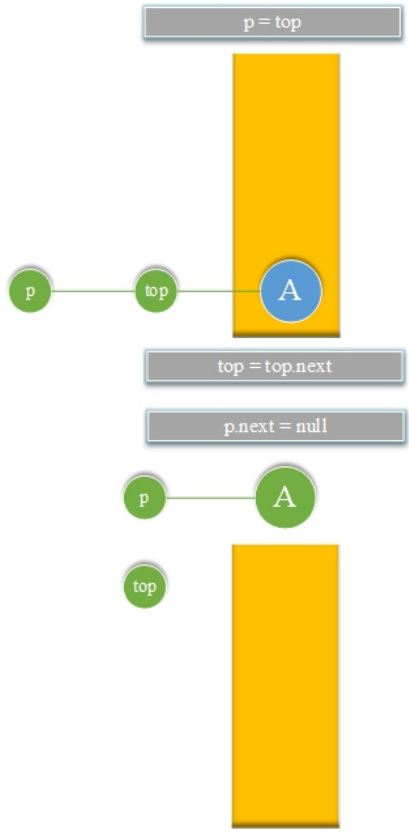
If pop C from Stack:



If pop B from Stack:



If pop A from Stack:



Stack.java

```
public class Stack {
    private Node top ;
    private int size ;
    public void push(String element) {
        if (top == null ) {
            top = new Node(element);
        } else {
            Node newNode = new Node(element);
            newNode.next = top ;
            top = newNode;
        }
        size ++;
    }
    public Node pop() {
        if (top == null )
        {
            return null ;
        }
        Node p = top ;
        top = top .next ;// top move down
        p.next = null ;
        size --;
        return p;
    }
    public int size() {
        return size ;
    }
}
```

TestStack.java

```
public class TestStack {
    public static void main(String[] args) {
        Stack stack = new Stack();
        stack.push("A" );
        stack.push("B" );
        stack.push("C" );
        print (stack);
    }
    public static void print(Stack stack) {
        System.out .print("Top " );
        Node node = null ;
        while ((node = stack.pop())!=null ) {
            System.out .print(node.data + " -> " );
        }
        System.out .print("End\n" );
    }
}
```

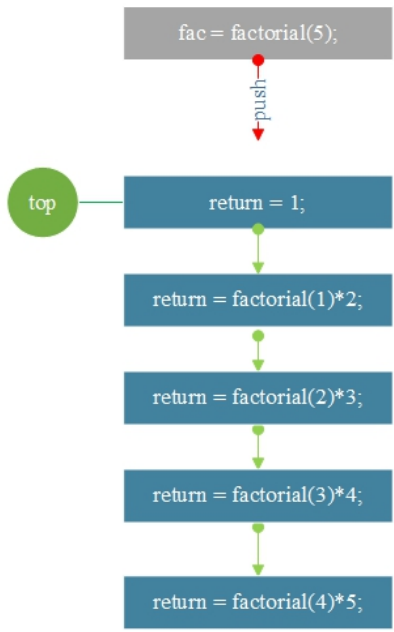
Result:

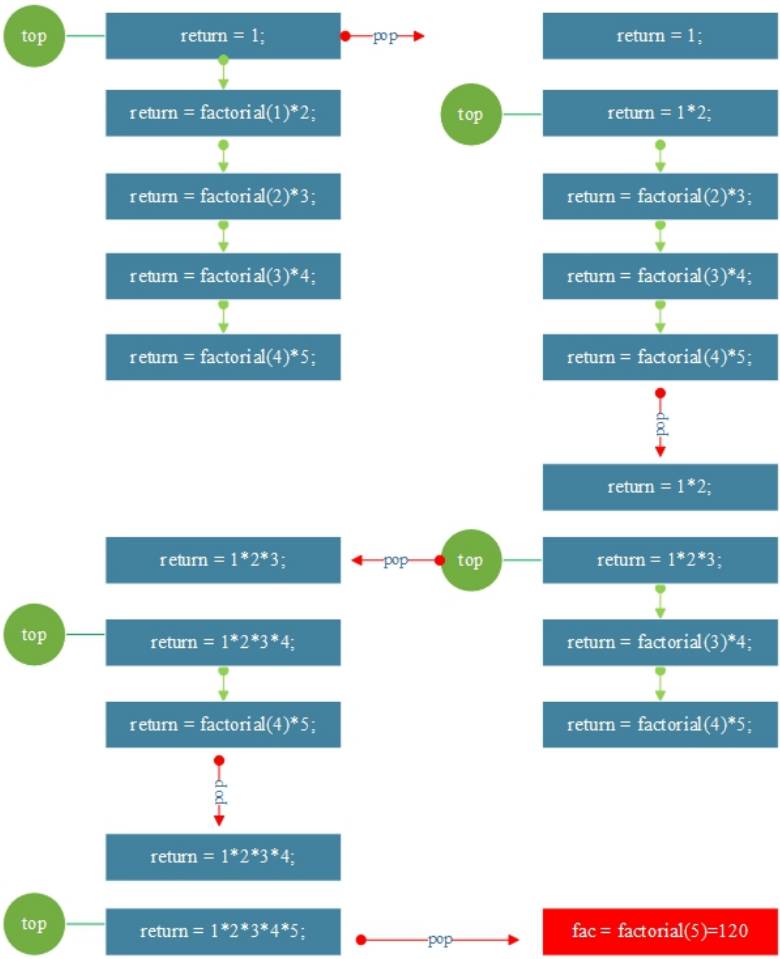
Top C -> B -> A -> End

Recursive Algorithm

Recursive Algorithm:
The program function itself calls its own to progress until it reaches a certain condition and step by step returns to the end..

1. Factorial of n : n*(n-1)*(n-2) *2*1




top
return = 1;
pop
return = 1;
return = factorial(1)*2;
top
return = 1*2;
return = factorial(2)*3;
return = factorial(2)*3;
return = factorial(3)*4;
return = factorial(3)*4;
return = factorial(4)*5;
return = factorial(4)*5;
pop
return = 1*2;
return = 1*2*3;
pop
top
return = 1*2*3;
top
return = 1*2*3*4;
return = factorial(3)*4;
return = factorial(4)*5;
return = factorial(4)*5;
pop
return = 1*2*3*4;
top
return = 1*2*3*4*5;
pop
fac = factorial(5)=120

TestFactorial.java

```
public class TestFactorial {
    public static void main(String[] args) {
        int n = 5;
        long fac = factorial (n);
        System.out .println("The factorial of 5 is :" + fac);
    }
    public static long factorial(int n) {
        if (n == 1) {
            return 1;
        } else {
            return factorial (n - 1) * n;//Recursively call yourself
until the end of the return
        }
    }
}
```

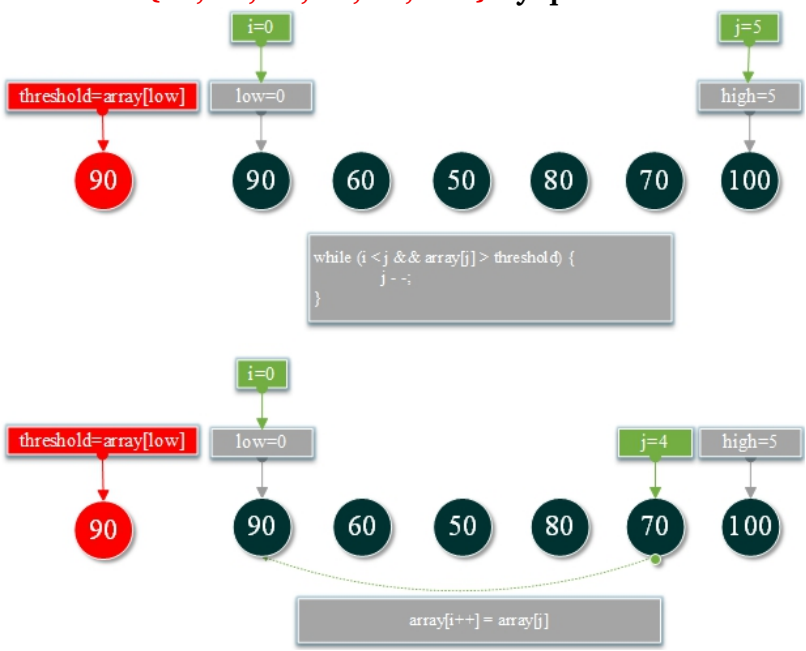
Result:

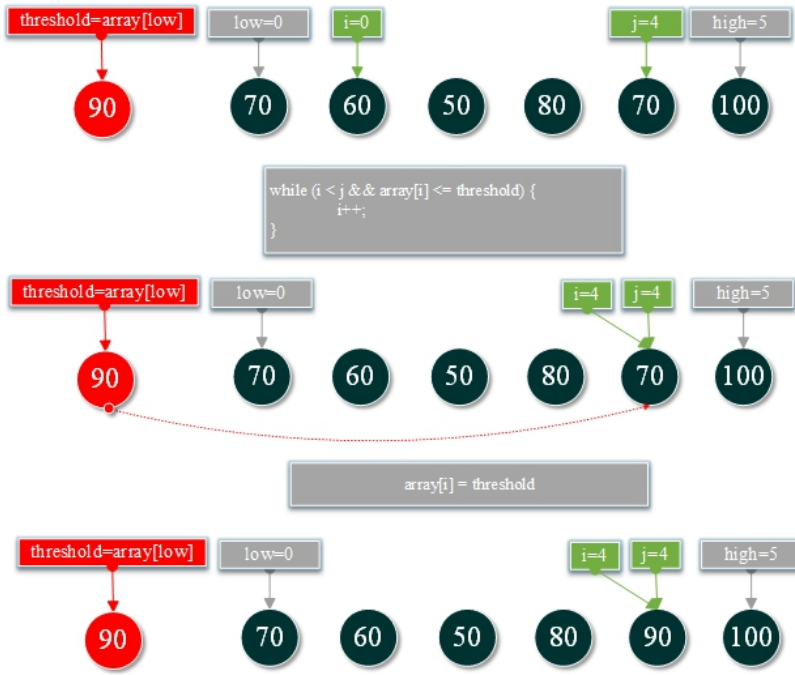
The factorial of 5 is :120

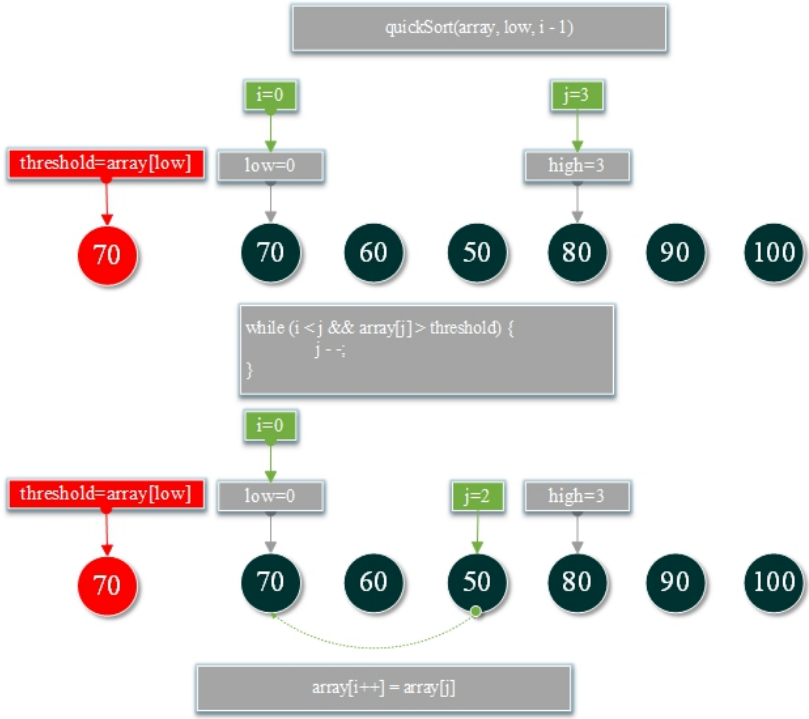
Quick Sort Algorithm

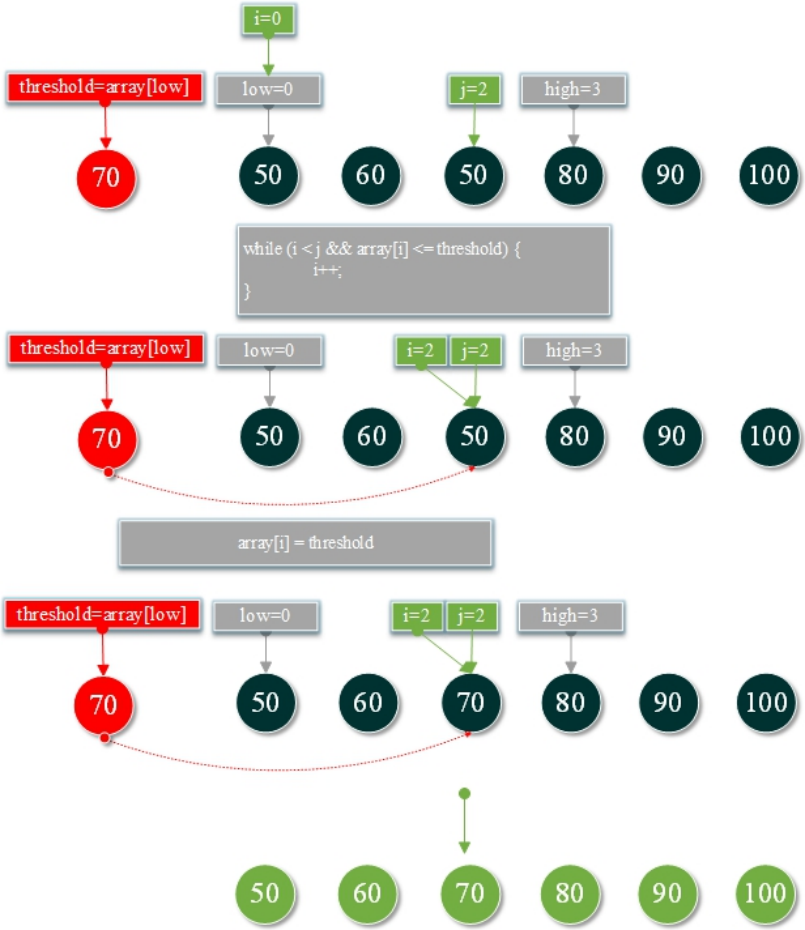
Quick Sort Algorithm:
Quicksort is a popular sorting algorithm that is often faster in practice compared to other sorting algorithms. It utilizes a divide-and-conquer strategy to quickly sort data items by dividing a large array into two smaller arrays.

1. The scores {90, 60, 50, 80, 70, 100} by quick sort




threshold=array[low]
low=0
i=0
j=4
high=5
90
70
60
50
80
70
100
while (i < j && array[i] <= threshold) {
i++;
}
threshold=array[low]
low=0
i=4
j=4
high=5
90
70
60
50
80
70
100
array[i] = threshold
threshold=array[low]
low=0
i=4
j=4
high=5
90
70
60
50
80
90
100


quickSort(array, low, i - 1)
i=0
j=3
threshold=array[low]
low=0
high=3
70
70
60
50
80
90
100
while (i < j && array[j] > threshold) {
j - -;
}
i=0
threshold=array[low]
low=0
j=2
high=3
70
70
60
50
80
90
100
array[i++] = array[j]


i=0
threshold=array[low]
low=0
j=2
high=3
70
50
60
50
80
90
100
while (i < j && array[i] <= threshold) {
i++;
}
threshold=array[low]
low=0
i=2
j=2
high=3
70
50
60
50
80
90
100
array[i] = threshold
threshold=array[low]
low=0
i=2
j=2
high=3
70
50
60
70
80
90
100
50
60
70
80
90
100

TestQuickSort.java

```
public class TestQuickSort {
    private static void quickSort(int [] array, int low, int high) {
        if (low > high) {
            return ;
        }
        int i = low;
        int j = high;
        int threshold = array[low];
        // Alternately scanned from both ends of the list
        while (i < j) {
            // Find the first position less than threshold from right to
left
            while (i < j && array[j] > threshold) {
                j--;
            }
            //Replace the low with a smaller number than the
threshold
            if (i < j)
                array[i++] = array[j];
            // Find the first position greater than threshold from left
to right
            while (i < j && array[i] <= threshold) {
                i++;
            }
            //Replace the high with a number larger than the
threshold
            if (i < j)
                array[j--] = array[i];
        }
        array[i] = threshold;
        quickSort (array, low, i - 1); // left quickSort
        quickSort (array, i + 1, high); // right quickSort
    }
    public static void quickSort(int [] array) {
        if (array.length > 0) {
            quickSort (array, 0, array.length - 1);
        }
    }
```

```
    public static void main(String[] args) {
        int [] scores = { 90, 60, 50, 80, 70, 100 };
        quickSort (scores);
        for (int i = 0; i < scores.length ; i++) {
            System.out .print(scores[i] + "," );
        }
    }
}
```

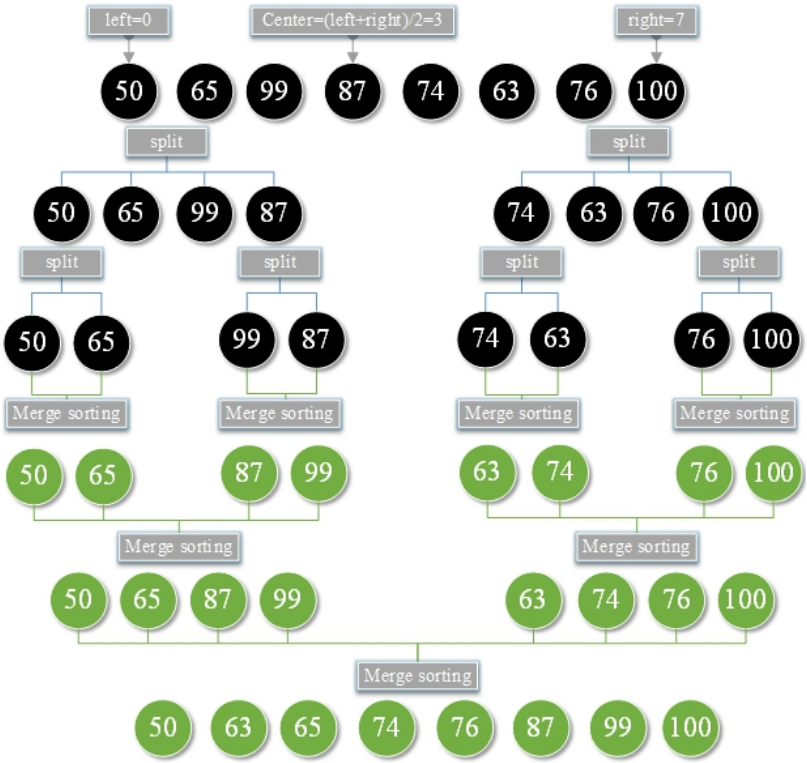
Result:

50,60,70,80,90,100

Two-way Merge Algorithm

Two-way Merge Algorithm:
The data of the first half and the second half are sorted, and the two ordered sub-list are merged into one ordered list, which continue to recursive to the end.

1. The scores {50, 65, 99, 87, 74, 63, 76, 100} by merge sort



TestMergeSort.java

```
public class TestMergeSort {
    public static void main(String args[]) {
        int scores[] = { 50, 65, 99, 87, 74, 63, 76, 100, 92 };
        mergeSort (scores);
        for (int i = 0; i < scores.length ; i++) {
            System.out .print(scores[i] + "," );
        }
    }
    private static void mergeSort(int [] array) {
        mergeSort (array, new int [array.length ], 0, array.length -
1);
    }
    private static void mergeSort(int [] array, int [] temp, int
left, int right) {
        if (left < right) {
            int center = (left + right) / 2;
            mergeSort (array, temp, left, center); // Left merge sort
            mergeSort (array, temp, center + 1, right); // Right
merge sort
            merge (array, temp, left, center + 1, right); // Merge two
ordered arrays
        }
    }
    /**
     Combine two ordered list into an ordered list
     temp : Temporary array
     left :    Start the subscript on the left
     right :  Start the subscript on the right
     rightEndIndex : End subscript on the right
     */
    private static void merge(int [] array, int [] temp, int left,
int right, int rightEndIndex) {
        int leftEndIndex = right - 1; // End subscript on the left
        int tempIndex = left; // Starting from the left count
        int elementNumber = rightEndIndex - left + 1;
        while (left <= leftEndIndex && right <= rightEndIndex)
{
            if (array[left] <= array[right])
```

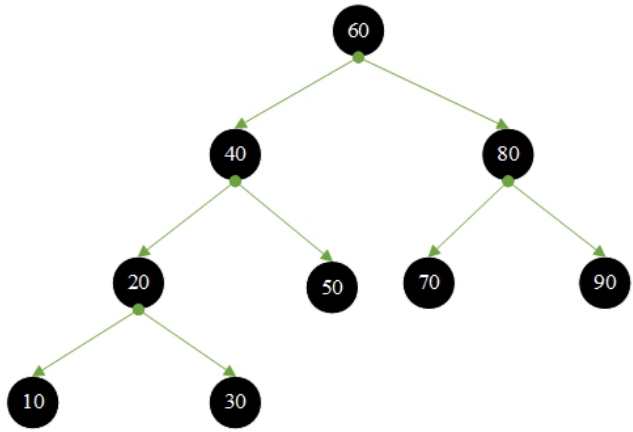
```
            temp[tempIndex++] = array[left++];
        else
            temp[tempIndex++] = array[right++];
    }
    while (left <= leftEndIndex) { // If there is element on
the left
        temp[tempIndex++] = array[left++];
    }
    while (right <= rightEndIndex) { // If there is element on
the right
        temp[tempIndex++] = array[right++];
    }
    // Copy temp to array
    for (int i = 0; i < elementNumber; i++) {
        array[rightEndIndex] = temp[rightEndIndex];
        rightEndIndex--;
    }
  }
}
```

Result:

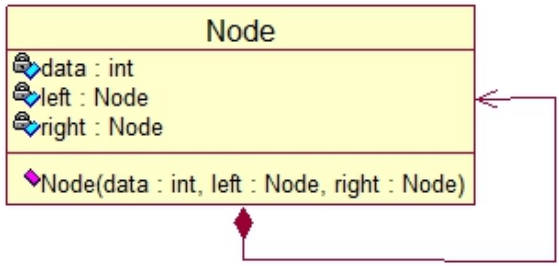
50,63,65,74,76,87,99,100

Binary Search Tree

Binary Search Tree:
1. If the left subtree of any node is not empty, the value of all nodes on the left subtree is less than the value of its root node;
2. If the right subtree of any node is not empty, the value of all nodes on the right subtree is greater than the value of its root node;
3. The left subtree and the right subtree of any node are also binary search trees.



Node UML Diagram



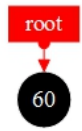
Node.java

```
public class Node {
    private int data ;
    public Node left ;
    public Node right ;
    public Node() {
    }
    public Node(int data, Node left, Node right) {
        this .data = data;
        this .left = left;
        this .right = right;
    }
    public int getData() {
        return data ;
    }
    public void setData(int data) {
        this .data = data;
    }
}
```

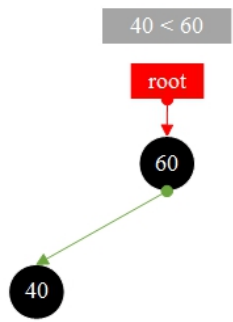
1. Construct a binary search tree, insert node

The inserted nodes are compared from the root node, and the smaller than the root node is compared with the left subtree of the root node, otherwise, compared with the right subtree until the left subtree is empty or the right subtree is empty, then is inserted.

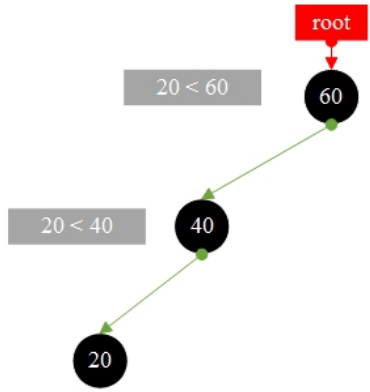
Insert 60



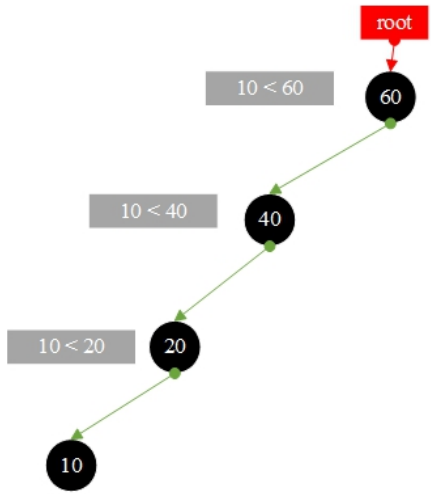
Insert 40



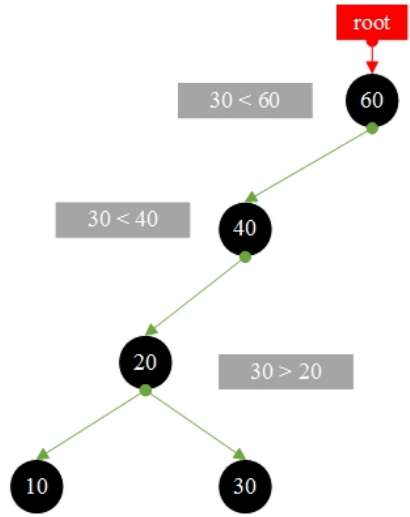
Insert 20



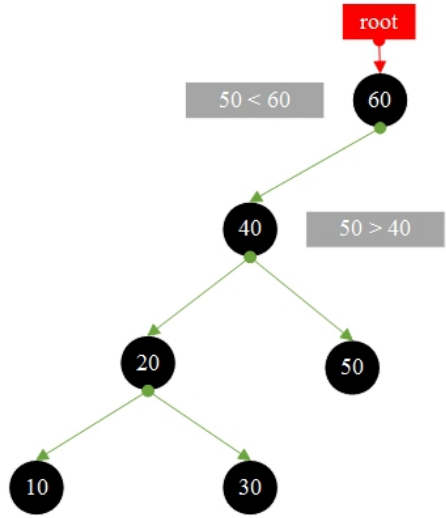
Insert 10



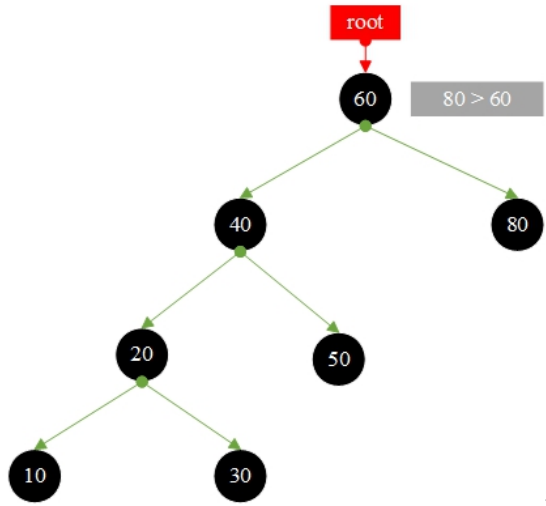
Insert 30



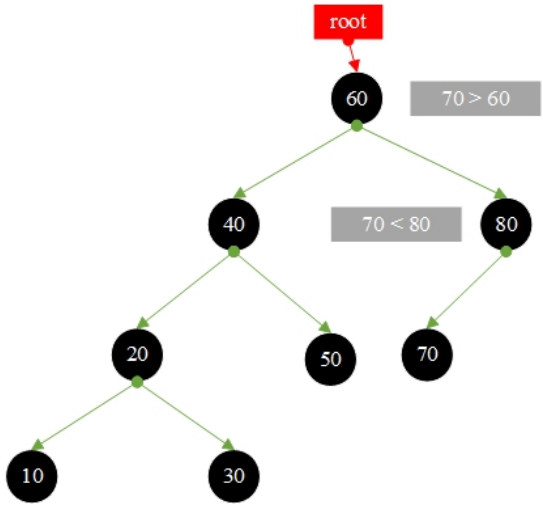
Insert 50



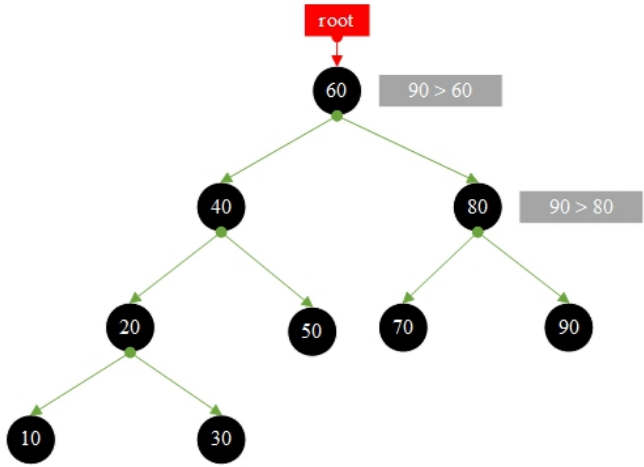
Insert 80



Insert 70

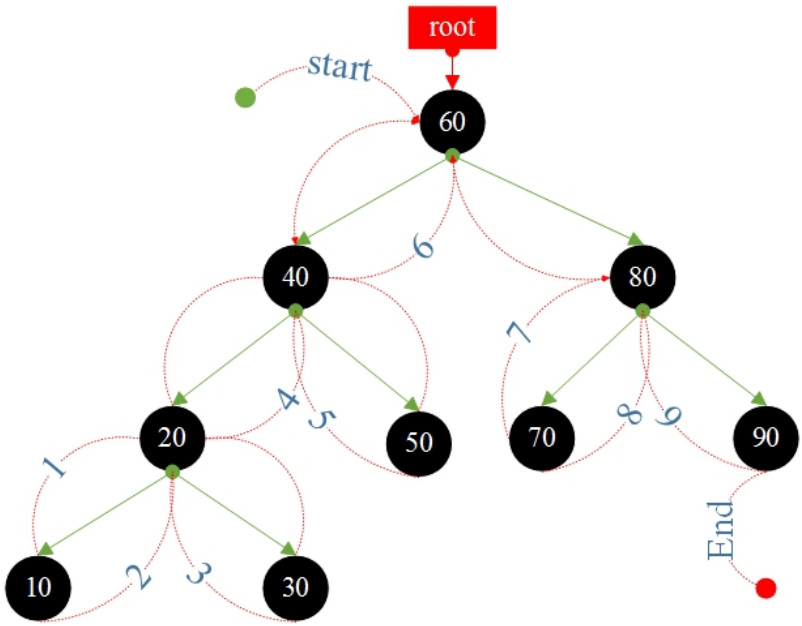


Insert 90

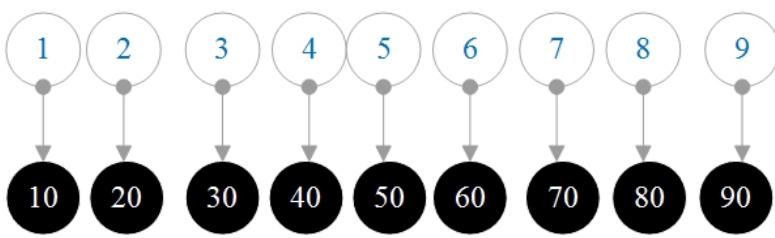


2. binary search tree In-order traversal

In-order traversal : left subtree -> root node -> right subtree



Result:



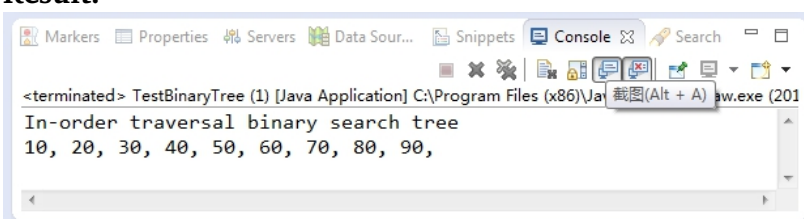
BinaryTree.java

```
public class BinaryTree {
    private Node root ;
    public Node getRoot() {
        return root ;
    }
    // In-order traversal binary search tree
    public void inOrder(Node root) {
        if (root == null ) {
            return ;
        }
        inOrder(root.left ); // Traversing the left subtree
        System.out .print(root.getData() + ", " );
        inOrder(root.right ); // Traversing the right subtree
    }
    public void insert(Node node, int newData) {
        if (this .root == null ) {
            this .root = new Node(newData, null , null );
            return ;
        }
        int compareValue = newData - node.getData();
        //Recursive left subtree, continue to find the insertion
position
        if (compareValue < 0) {
            if (node.left == null ) {
                node.left = new Node(newData, null , null );
            } else {
                insert(node.left , newData);
            }
        } else if (compareValue > 0) {////Recursive right subtree
to find the insertion position
            if (node.right == null ) {
                node.right = new Node(newData, null , null );
            } else {
                insert(node.right , newData);
            }
        }
    }
}
```

TestBinaryTree.java

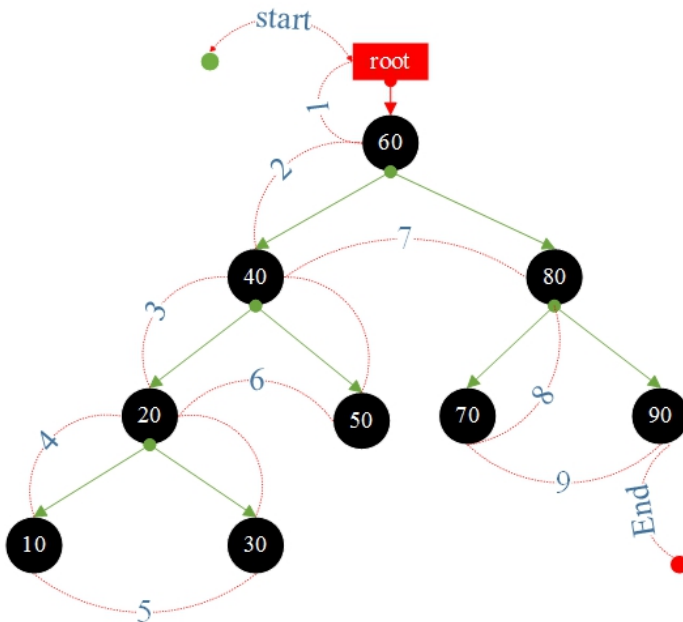
```
public class TestBinaryTree {
    public static void main(String[] args) {
        BinaryTree binaryTree = new BinaryTree();
        //Constructing a binary search tree
        binaryTree.insert(binaryTree.getRoot(), 60);
        binaryTree.insert(binaryTree.getRoot(), 40);
        binaryTree.insert(binaryTree.getRoot(), 20);
        binaryTree.insert(binaryTree.getRoot(), 10);
        binaryTree.insert(binaryTree.getRoot(), 30);
        binaryTree.insert(binaryTree.getRoot(), 50);
        binaryTree.insert(binaryTree.getRoot(), 80);
        binaryTree.insert(binaryTree.getRoot(), 70);
        binaryTree.insert(binaryTree.getRoot(), 90);
        System.out .println("\nIn-order traversal binary search tree"
);
        binaryTree.inOrder(binaryTree.getRoot());
    }
}
```

Result:

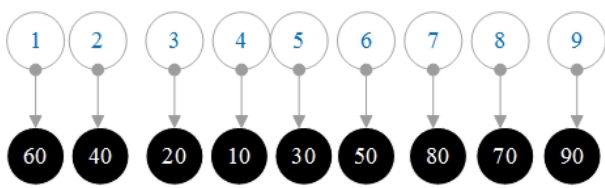


3. binary search tree Pre-order traversal

Pre-order traversal : root node -> left subtree -> right subtree



Result:



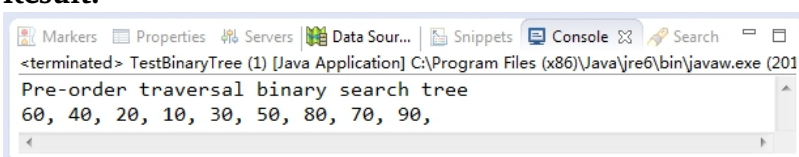
BinaryTree.java

```
public class BinaryTree {
    private Node root ;
    public Node getRoot() {
        return root ;
    }
    // Preorder traversal binary search tree
    public void preOrder(Node  root) {
        if (root == null ) {
            return ;
        }
        System.out .print(root.getData() + ", " );
        preOrder(root.left ); // Recursive Traversing the left subtree
        preOrder(root.right ); // Recursive Traversing the right
subtree
    }
    public void insert(Node node, int newData) {
        if (this .root == null ) {
            this .root = new Node(newData, null , null );
            return ;
        }
        int compareValue = newData - node.getData();
        //Recursive left subtree, continue to find the insertion
position
        if (compareValue < 0) {
            if (node.left == null ) {
                node.left = new Node(newData, null , null );
            } else {
                insert(node.left , newData);
            }
        } else if (compareValue > 0) {
            if (node.right == null ) {
                node.right = new Node(newData, null , null );
            } else {
                insert(node.right , newData);
            }
        }
    }
}
```

TestBinaryTree.java

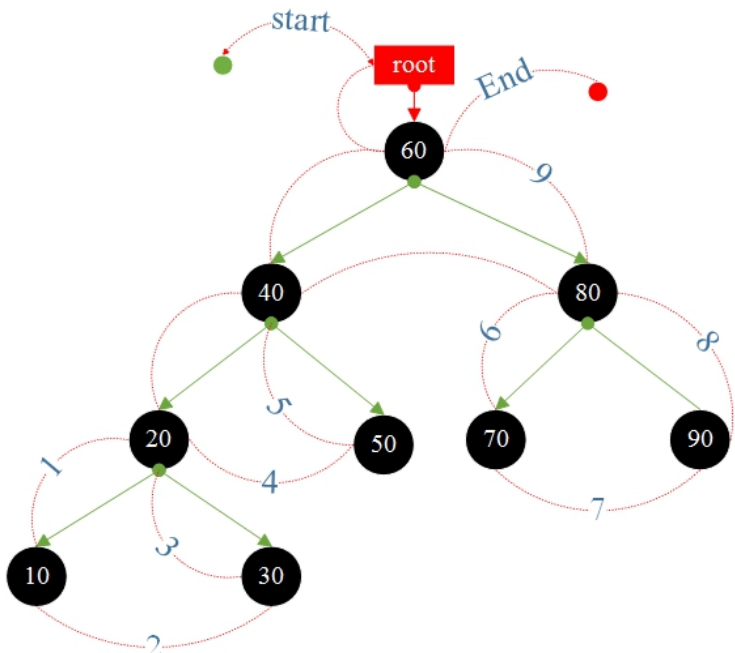
```
public class TestBinaryTree {
    public static void main(String[] args) {
        BinaryTree binaryTree = new BinaryTree();
        //Constructing a binary search tree
        binaryTree.insert(binaryTree.getRoot(), 60);
        binaryTree.insert(binaryTree.getRoot(), 40);
        binaryTree.insert(binaryTree.getRoot(), 20);
        binaryTree.insert(binaryTree.getRoot(), 10);
        binaryTree.insert(binaryTree.getRoot(), 30);
        binaryTree.insert(binaryTree.getRoot(), 50);
        binaryTree.insert(binaryTree.getRoot(), 80);
        binaryTree.insert(binaryTree.getRoot(), 70);
        binaryTree.insert(binaryTree.getRoot(), 90);
        System.out .println("Pre-order traversal binary search tree"
);
        binaryTree.preOrder(binaryTree.getRoot());
    }
}
```

Result:

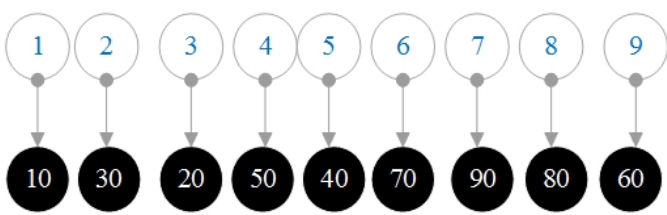


4. binary search tree Post-order traversal

Post-order traversal : right subtree -> root node -> left subtree



Result:



BinaryTree.java

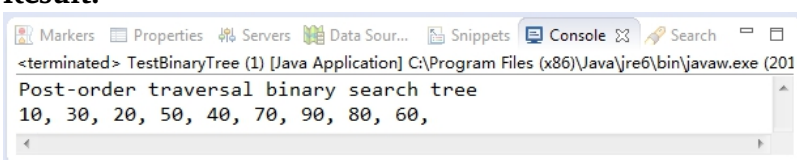
```
public class BinaryTree {
    private Node root ;
    public Node getRoot() {
        return root ;
    }
    //Post-order traversal binary search tree
    public void postOrder(Node root) {
        if (root == null ) {
            return ;
        }
        postOrder(root.left ); // Recursive Traversing the left
subtree
        postOrder(root.right ); // Recursive Traversing the right
subtree
        System.out .print(root.getData() + ", " );
    }
    public void insert(Node node, int newData) {
        if (this .root == null ) {
            this .root = new Node(newData, null , null );
            return ;
        }
        int compareValue = newData - node.getData();
        //Recursive left subtree, continue to find the insertion
position
        if (compareValue < 0) {
            if (node.left == null ) {
                node.left = new Node(newData, null , null );
            } else {
                insert(node.left , newData);
            }
        } else if (compareValue > 0) {//Recursive right subtree to
find the insertion position
            if (node.right == null ) {
                node.right = new Node(newData, null , null );
            } else {
                insert(node.right , newData);
            }
        }
```

```
    }
}
```

TestBinaryTree.java

```
public class TestBinaryTree {
    public static void main(String[] args) {
        BinaryTree binaryTree = new BinaryTree();
        //Constructing a binary search tree
        binaryTree.insert(binaryTree.getRoot(), 60);
        binaryTree.insert(binaryTree.getRoot(), 40);
        binaryTree.insert(binaryTree.getRoot(), 20);
        binaryTree.insert(binaryTree.getRoot(), 10);
        binaryTree.insert(binaryTree.getRoot(), 30);
        binaryTree.insert(binaryTree.getRoot(), 50);
        binaryTree.insert(binaryTree.getRoot(), 80);
        binaryTree.insert(binaryTree.getRoot(), 70);
        binaryTree.insert(binaryTree.getRoot(), 90);
        System.out .println("Post-order traversal binary search tree"
);
        binaryTree.postOrder(binaryTree.getRoot());
    }
}
```

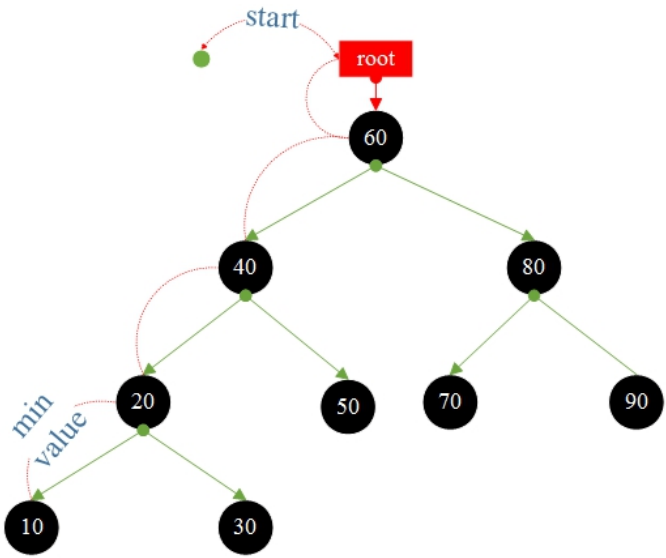
Result:

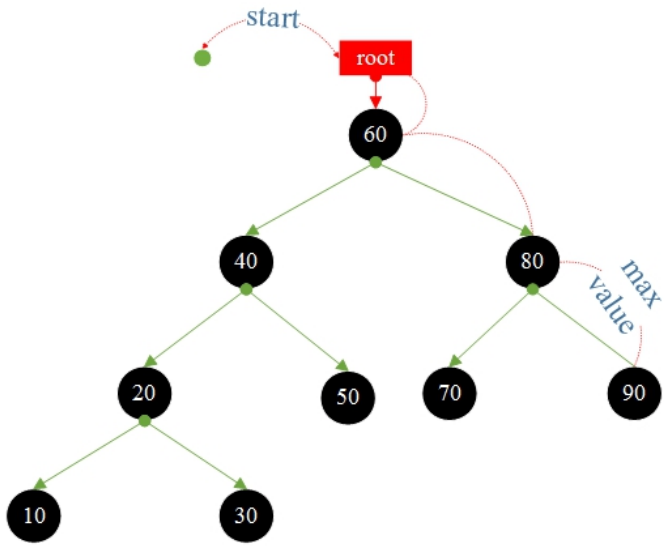


5. binary search tree Maximum and minimum

Minimum value: The small value is on the left child node, as long as the recursion traverses the left child until be empty, the current node is the minimum node.

Maximum value: The large value is on the right child node, as long as the recursive traversal is the right child until be empty, the current node is the largest node.




start
root
60
40
80
max
value
20
50
70
90
10
30

BinaryTree.java

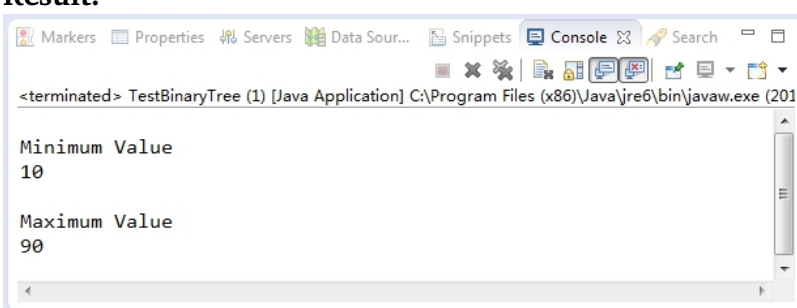
```
public class BinaryTree {
    private Node root ;
    public Node getRoot() {
        return root ;
    }
    //Minimum value
    public Node searchMinValue(Node node) {
        if (node == null || node.getData() == 0)
            return null ;
        if (node.left == null ) {
            return node;
        }
        return searchMinValue(node.left );//Recursively find the
minimum from the left subtree
    }
    //Maximum value
    public Node searchMaxValue(Node node) {
        if (node == null || node.getData() == 0)
            return null ;
        if (node.right == null ) {
            return node;
        }
        return searchMaxValue(node.right );//Recursively find the
minimum from the right subtree
    }
    public void insert(Node node, int newData) {
        if (this .root == null ) {
            this .root = new Node(newData, null , null );
            return ;
        }
        int compareValue = newData - node.getData();
        //Recursive left subtree, continue to find the insertion
position
        if (compareValue < 0) {
            if (node.left == null ) {
                node.left = new Node(newData, null , null );
            } else {
                insert(node.left , newData);
```

```
            }
        } else if (compareValue > 0) {//Recursive right subtree to
find the insertion position
            if (node.right == null ) {
                node.right = new Node(newData, null , null );
            } else {
                insert(node.right , newData);
            }
        }
    }
}
```

TestBinaryTree.java

```
public class TestBinaryTree {
    public static void main(String[] args) {
        BinaryTree binaryTree = new BinaryTree();
        //Constructing a binary search tree
        binaryTree.insert(binaryTree.getRoot(), 60);
        binaryTree.insert(binaryTree.getRoot(), 40);
        binaryTree.insert(binaryTree.getRoot(), 20);
        binaryTree.insert(binaryTree.getRoot(), 10);
        binaryTree.insert(binaryTree.getRoot(), 30);
        binaryTree.insert(binaryTree.getRoot(), 50);
        binaryTree.insert(binaryTree.getRoot(), 80);
        binaryTree.insert(binaryTree.getRoot(), 70);
        binaryTree.insert(binaryTree.getRoot(), 90);
        System.out .println("\nMinimum Value" );
        Node
minNode = binaryTree.searchMinValue(binaryTree.getRoot());
        System.out .println(minNode.getData());
        System.out .println("\nMaximum Value" );
        Node
maxNode = binaryTree.searchMaxValue(binaryTree.getRoot());
        System.out .println(maxNode.getData());
    }
}
```

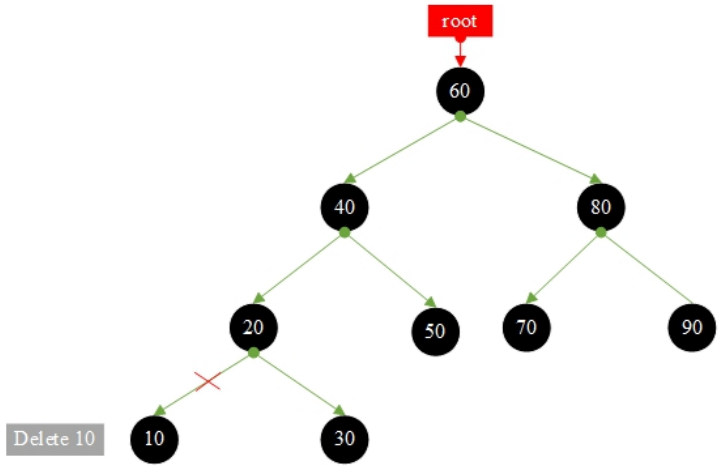
Result:

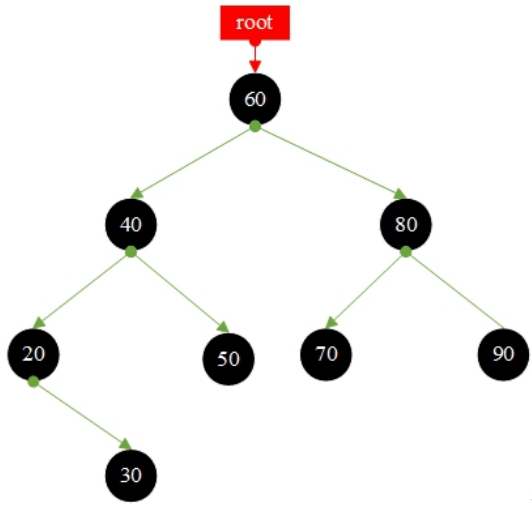


6. binary search tree Delete Node

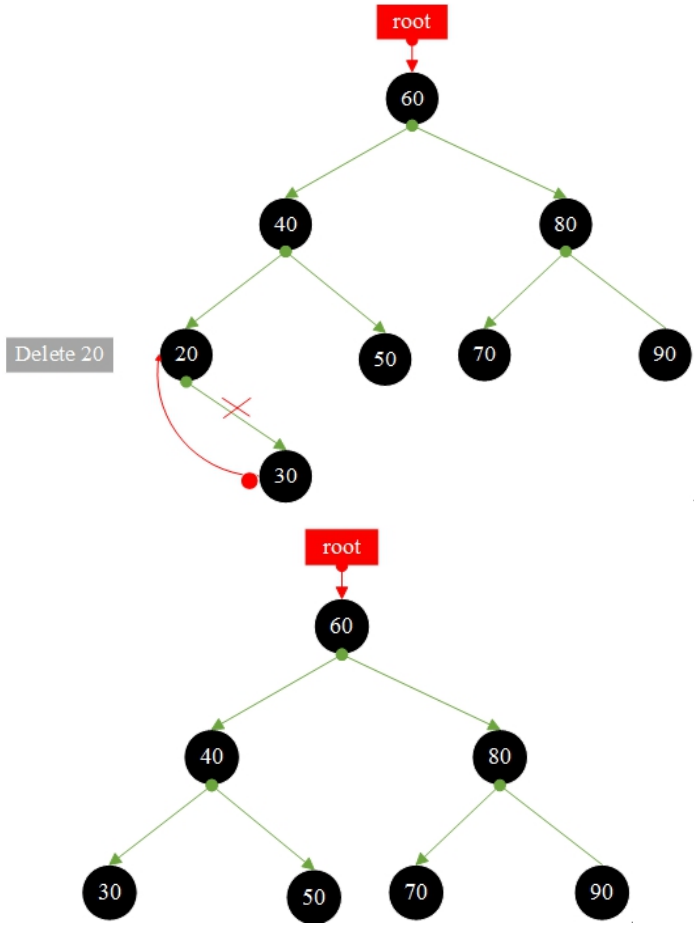
Binary search tree delete node 3 cases
1. If there is no child node, delete it directly
2. If there is only one child node, the child node replaces the current node, and then deletes the current node.
3. If there are two child nodes, replace the current node with the smallest node from the right subtree, because the smallest node on the right is also larger than the value on the left.

1. If there is no child node, delete it directly: delete node 10

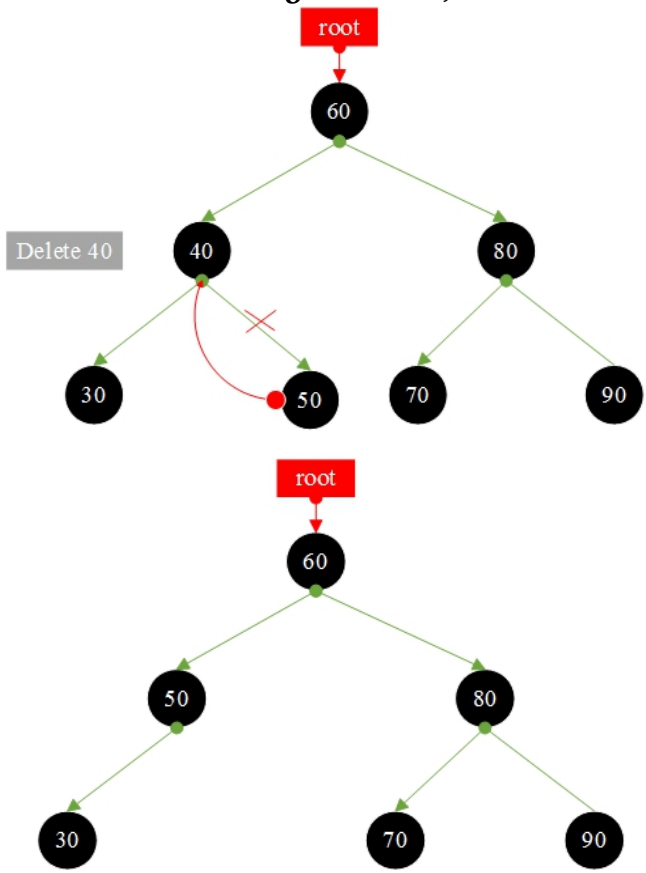



root
60
40
80
20
50
70
90
30

2. If there is only one child node, the child node replaces the current node, and then deletes the current node. Delete node 20



3. If there are two child nodes, replace the current node with the smallest node from the right subtree, Delete node 40

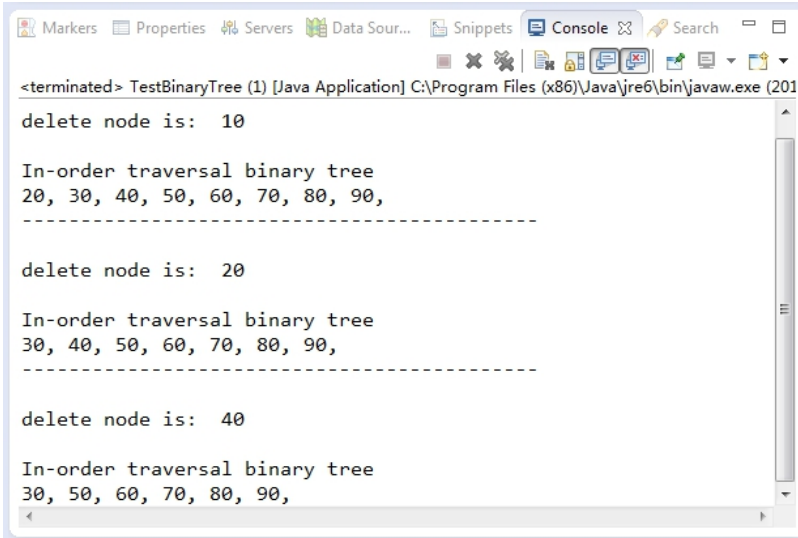


BinaryTree.java

TestBinaryTree.java

```
public class TestBinaryTree {
    public static void main(String[] args) {
        BinaryTree binaryTree = new BinaryTree();
        //Constructing a binary search tree
        binaryTree.insert(binaryTree.getRoot(), 60);
        binaryTree.insert(binaryTree.getRoot(), 40);
        binaryTree.insert(binaryTree.getRoot(), 20);
        binaryTree.insert(binaryTree.getRoot(), 10);
        binaryTree.insert(binaryTree.getRoot(), 30);
        binaryTree.insert(binaryTree.getRoot(), 50);
        binaryTree.insert(binaryTree.getRoot(), 80);
        binaryTree.insert(binaryTree.getRoot(), 70);
        binaryTree.insert(binaryTree.getRoot(), 90);
        System.out .println("\ndelete node is:  10" );
        binaryTree.remove(binaryTree.getRoot(), 10);
        System.out .println("\nIn-order traversal binary tree" );
        binaryTree.inOrder(binaryTree.getRoot());
        System.out .println("\n----------------------------------------" );
        System.out .println("\ndelete node is:  20" );
        binaryTree.remove(binaryTree.getRoot(), 20);
        System.out .println("\nIn-order traversal binary tree" );
        binaryTree.inOrder(binaryTree.getRoot());
        System.out .println("\n----------------------------------------" );
        System.out .println("\ndelete node is:  40" );
        binaryTree.remove(binaryTree.getRoot(), 40);
        System.out .println("\nIn-order traversal binary tree" );
        binaryTree.inOrder(binaryTree.getRoot());
    }
}
```

Result:


Markers
Properties
Servers
Data Sour...
Snippets
Console
Search
<terminated> TestBinaryTree (1) [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (201
delete node is: 10
In-order traversal binary tree
20, 30, 40, 50, 60, 70, 80, 90,
delete node is: 20
In-order traversal binary tree
30, 40, 50, 60, 70, 80, 90,
delete node is: 40
In-order traversal binary tree
30, 50, 60, 70, 80, 90,

Binary Heap Sorting

Binary Heap Sorting:
The value of the non-terminal node in the binary tree is not greater than the value of its left and right child nodes.

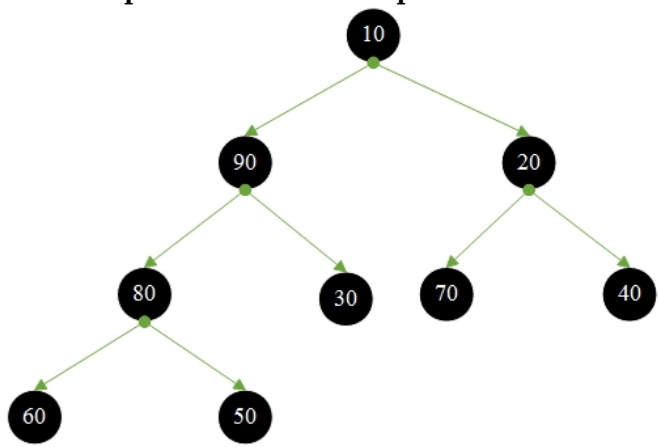
Small top heap : ki $<=$ k2i and ki $<=$ k2i+1
Big top heap :ki $>=$ k2i and ki $>=$ k2i+1

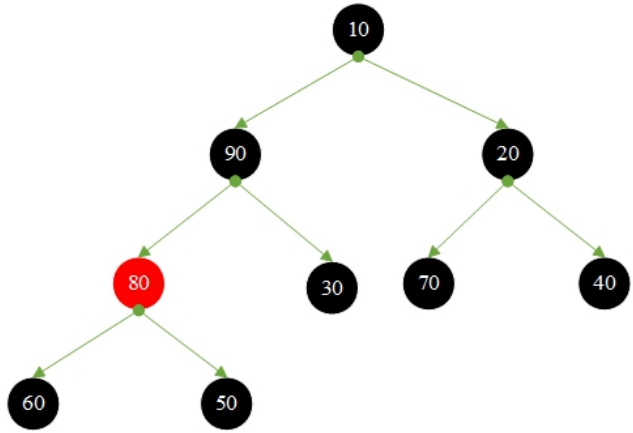
Parent node subscript = (i-1)/2
Left subnode subscript = 2*i+1
Right subnode subscript = 2*i+2

Heap sorting process:
1. Build a heap
2. After outputting the top element of the heap, adjust from top to bottom, compare the top element with the root node of its left and right subtrees, and swap the smallest element to the top of the heap; then adjust continuously until the leaf nodes to get new heap.

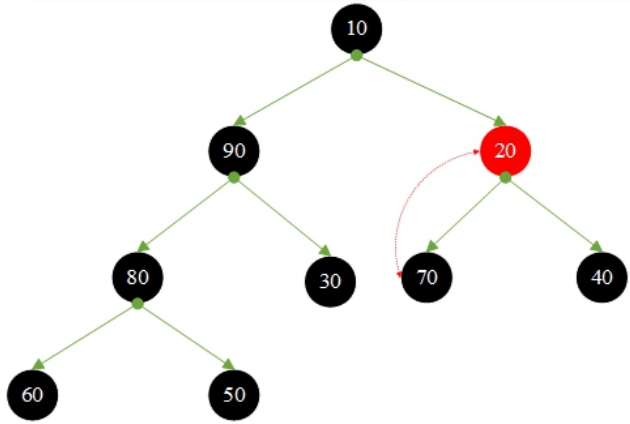
1. {10, 90, 20, 80, 30, 70, 40, 60, 50} build heap and then heap sort output.

Initialize the heap and build the heap

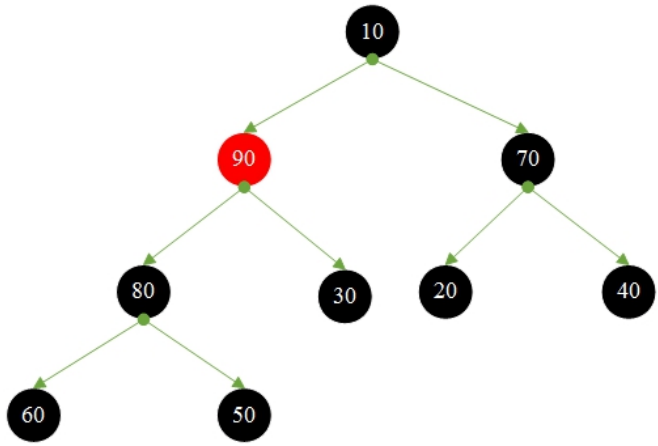




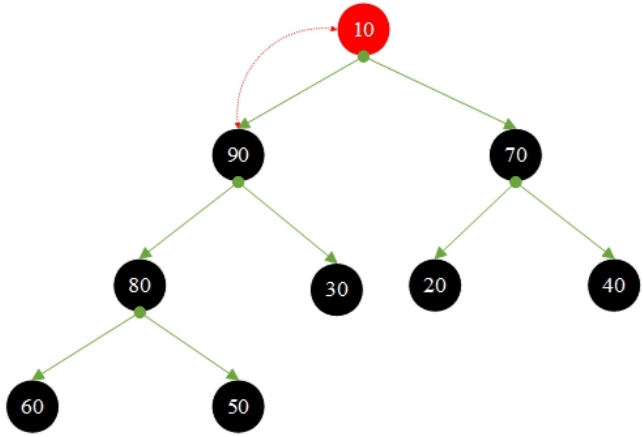
Not Leaf Node = 80 > left = 60 , 80 > right = 50 No need to move



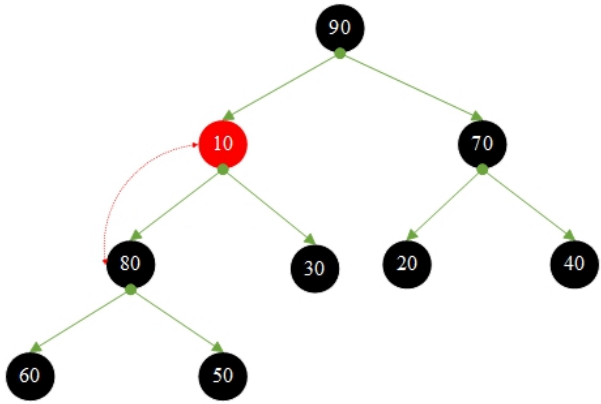
Not Leaf Node = 20 < left = 70 , 70 > right = 40 , 20 swap with 70



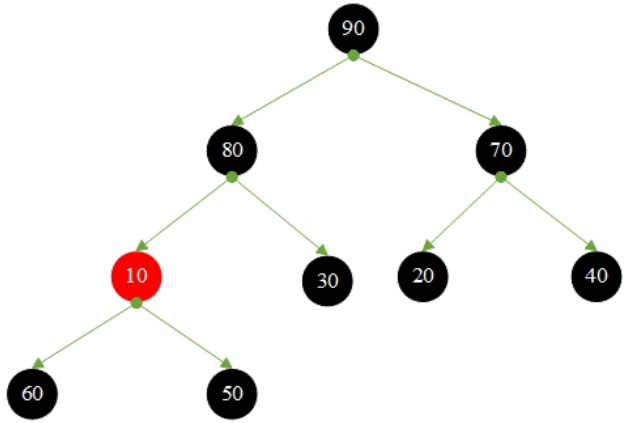
Not Leaf Node = 90 > left = 80 ， 80 > right = 30 No need to move



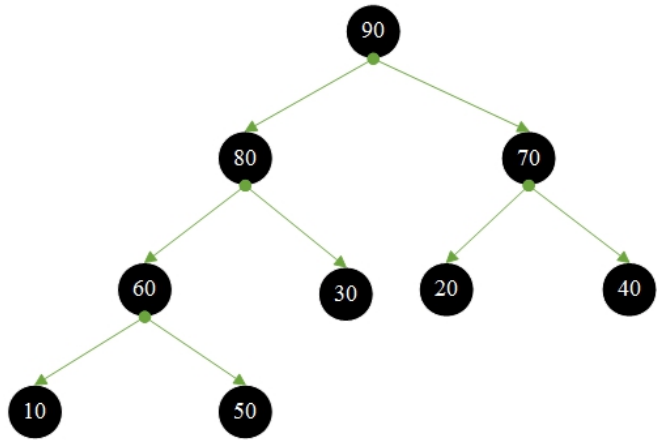
Not Leaf Node = 10 < left = 90 , 90 > right = 70 ， 10 swap with 90



Still Not Leaf Node = 10 < left = 80 , 80 > right =30 , 10 swap with 80

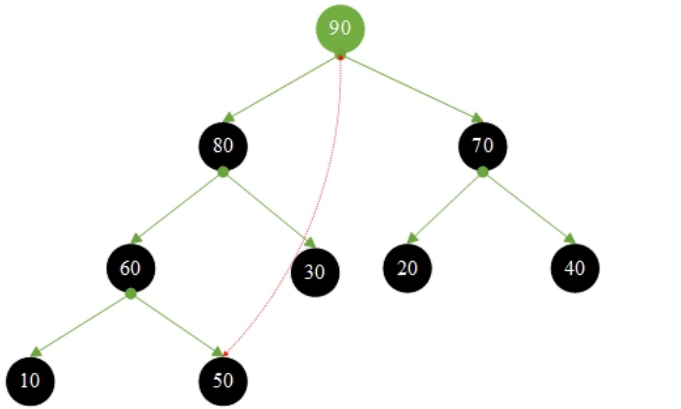


Still Not Leaf Node = 10 < left = 60 , 60 > right =50 , 10 swap with 60

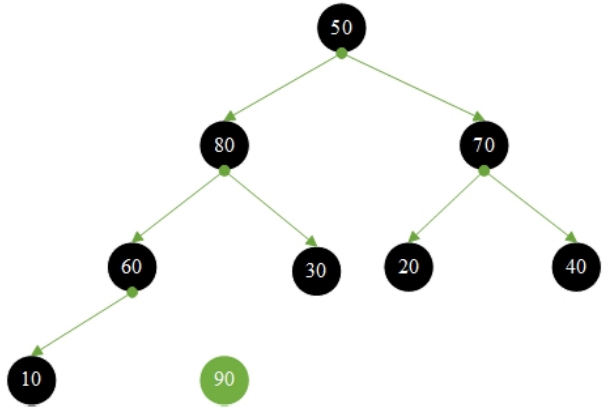


Create the heap finished

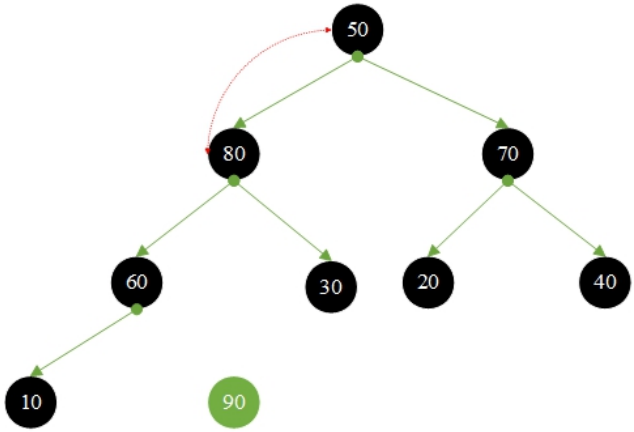
2. Start heap sorting

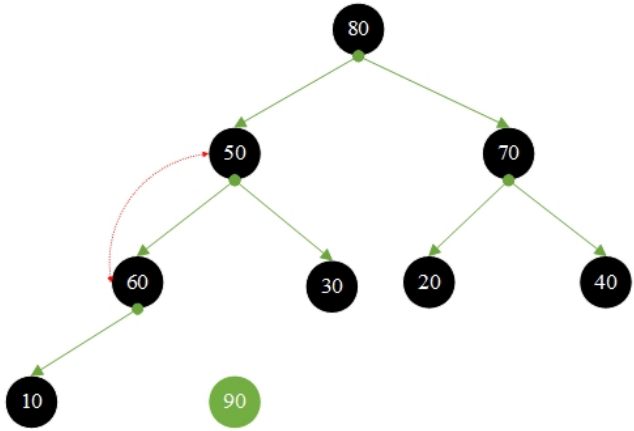


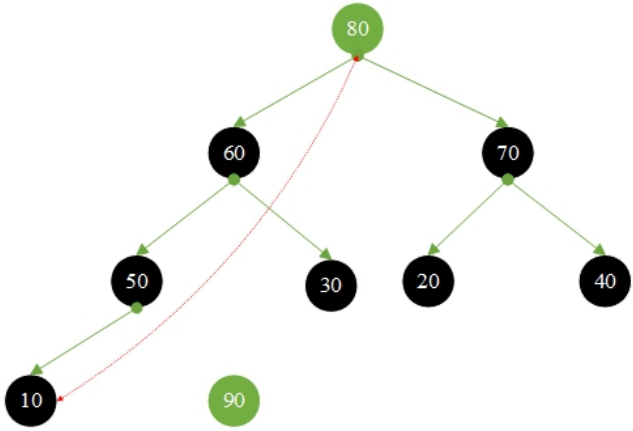
root = 90 and tail = 50 are exchanged



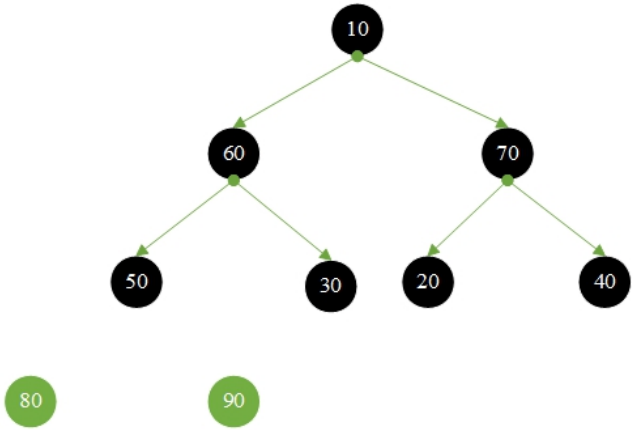
adjust the heap



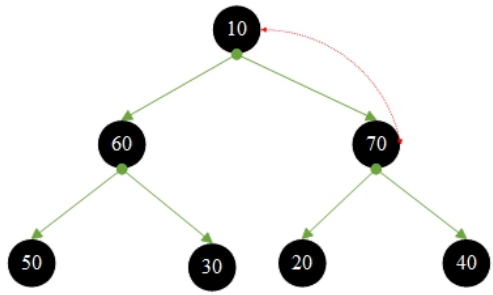




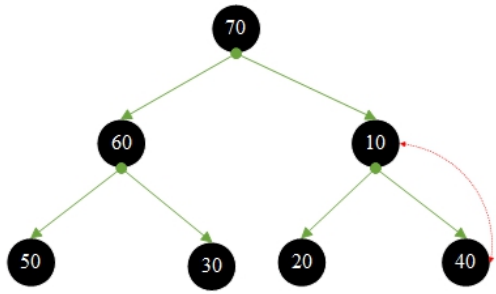
root = 80 and tail = 10 are exchanged



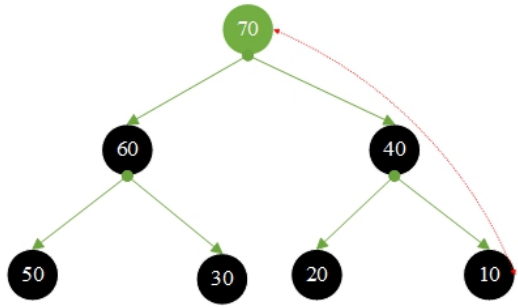
adjust the heap



80 90

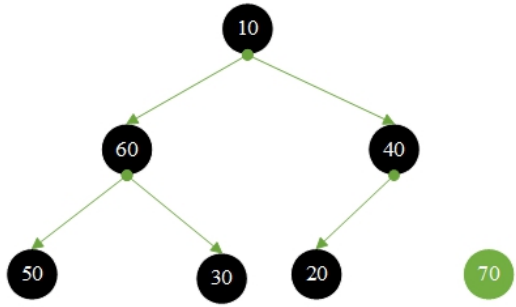


80 90



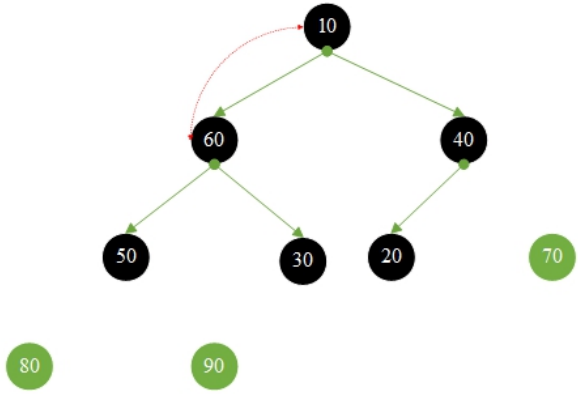
80 90

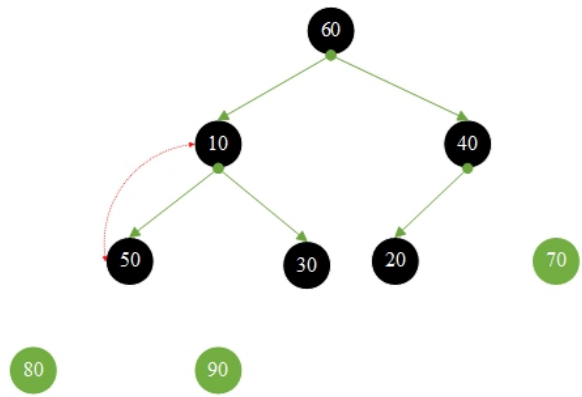
root = 70 and tail = 10 are exchanged

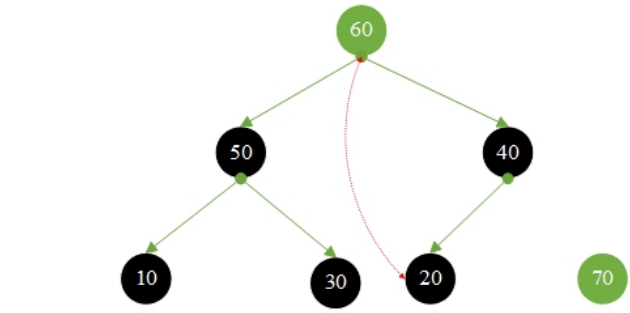


80 90

adjust the heap

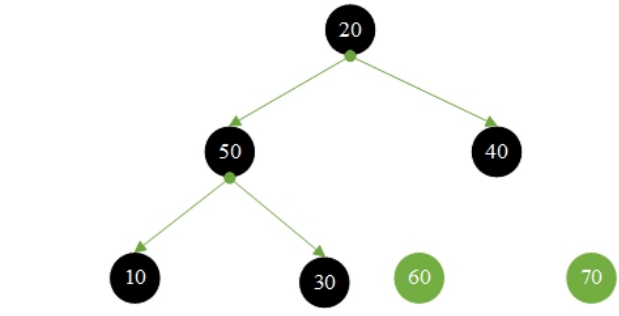






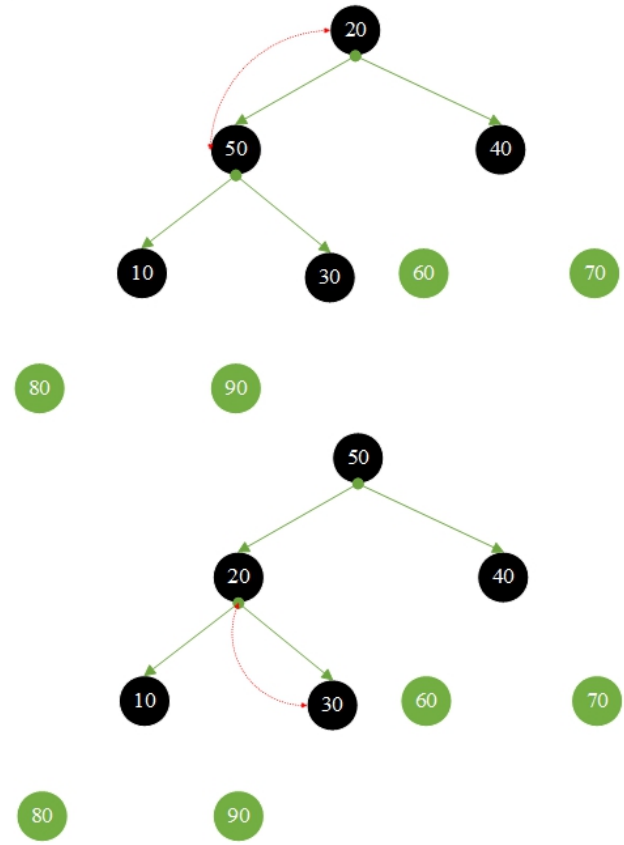
80 90

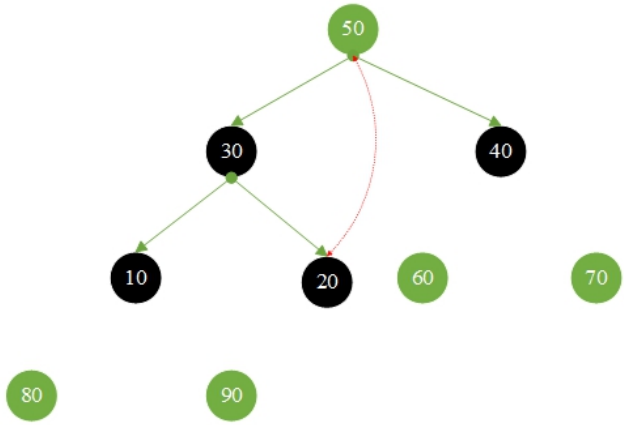
root = 60 and tail = 20 are exchanged



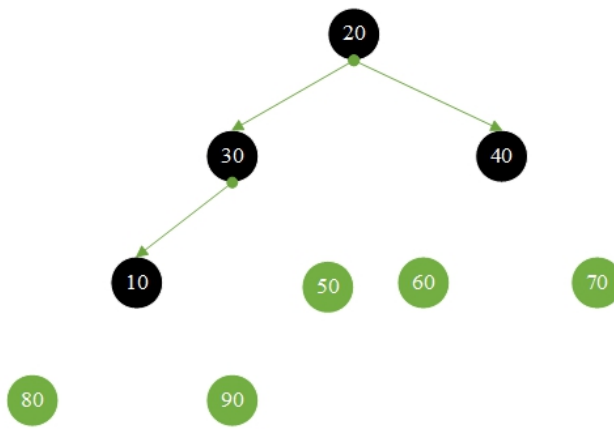
80 90

adjust the heap

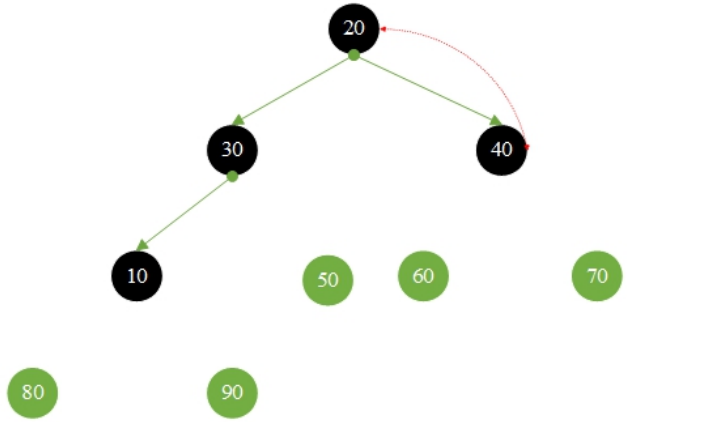


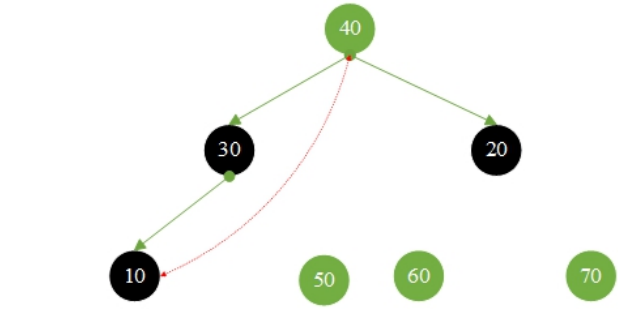


root = 50 and tail = 20 are exchanged



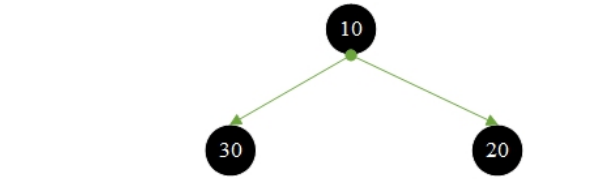
adjust the heap





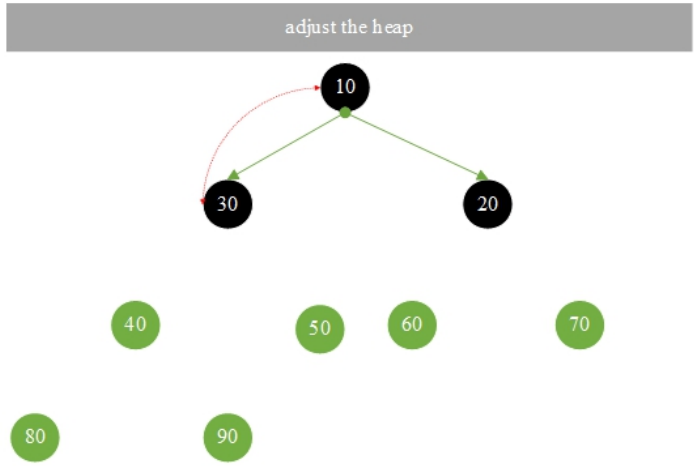
80 90

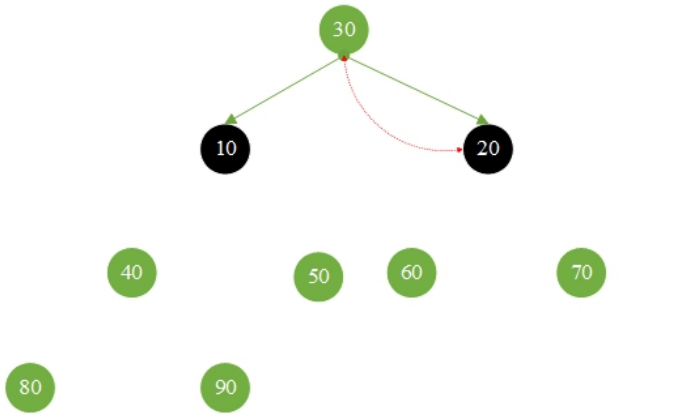
root = 40 and tail = 10 are exchanged



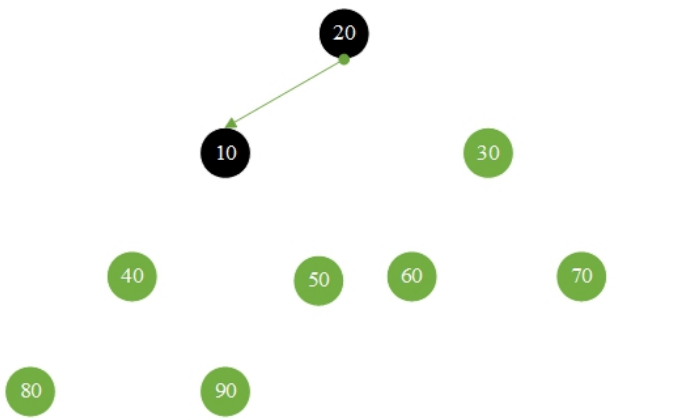
40 50 60 70

80 90

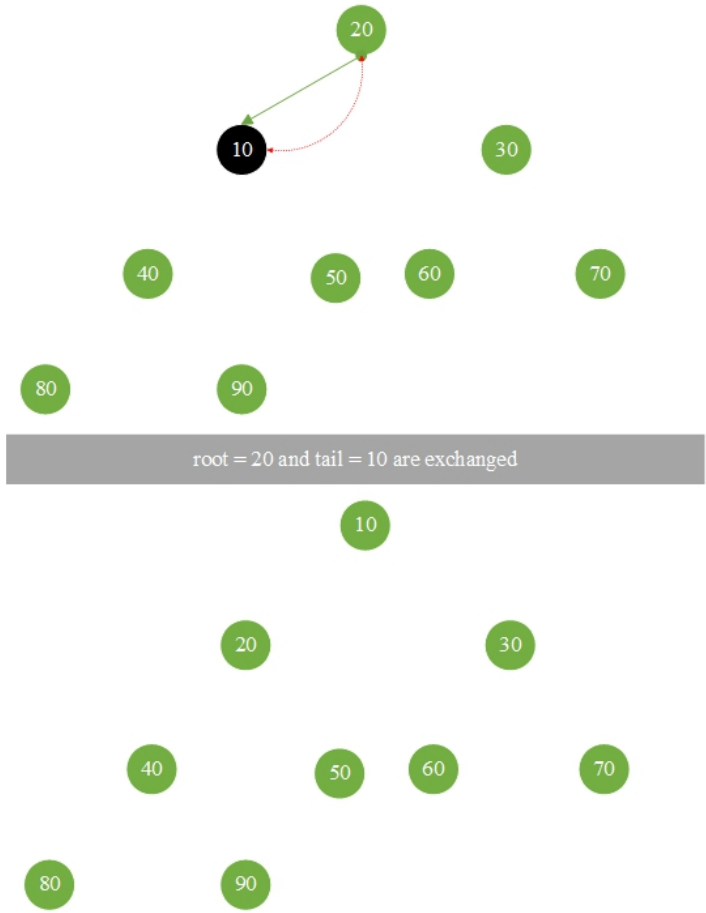

adjust the heap
10
30
20
40
50
60
70
80
90



root = 30 and tail = 20 are exchanged



no need adjust the heap


20
10
30
40
50
60
70
80
90
root = 20 and tail = 10 are exchanged
10
20
30
40
50
60
70
80
90

Heap sort result

HeapSort.java

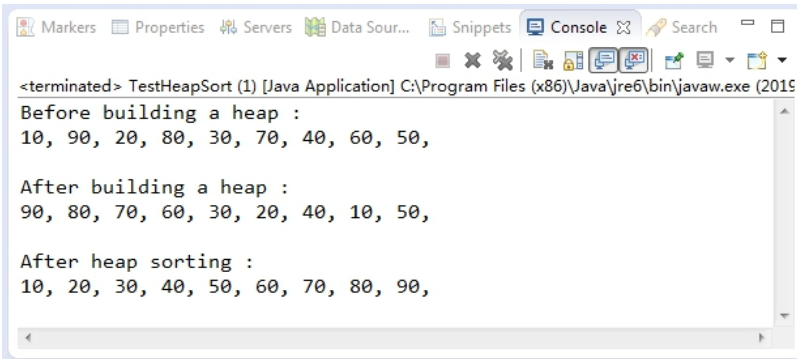
```
public class HeapSort {
    private int [] array ;
    //Initialize the heap
    public void createHeap(int [] array) {
        this .array = array;
        // Build a heap, (array.length - 1) / 2 scan half of the nodes
with child nodes
        for (int i = (array.length - 1) / 2; i >= 0; i--) {
            adjustHeap(i, array.length - 1);
        }
    }
    //Adjustment heap
    public void adjustHeap(int currentIndex, int maxLength) {
        int noLeafValue = array [currentIndex]; // Current non-
leaf node
        //2 * currentIndex + 1  Current left subtree subscript
        for (int j = 2 * currentIndex + 1; j <= maxLength; j =
currentIndex * 2 + 1) {
            if (j < maxLength && array [j] < array [j + 1]) {
                j++; // j Large subscript
            }
            if (noLeafValue >= array [j]) {
                break ;
            }
            array [currentIndex] = array [j]; // Move up to the
parent node
            currentIndex = j;
        }
        array [currentIndex] = noLeafValue; // To put in the
position
    }
    public void heapSort() {
        for (int i = array .length - 1; i > 0; i--) {
            int temp = array [0];
```

```
            array [0] = array [i];
            array [i] = temp;
            adjustHeap(0, i - 1);
        }
    }
}
```

TestHeapSort.java

```
public class TestHeapSort {
    public static void main(String[] args) {
        HeapSort heapSort = new HeapSort();
        int [] scores = { 10, 90, 20, 80, 30, 70, 40, 60, 50 };
        System.out .println("Before building a heap : " );
        for (int i = 0; i < scores.length ; i++ ) {
            System.out .print(scores[i] + ", " );
        }
        System.out .print("\n\n" );
        ////////////////////////////////////////////////////////////
        System.out .println("After building a heap : " );
        heapSort.createHeap(scores);
        for (int i = 0; i < scores.length ; i++ ) {
            System.out .print(scores[i] + ", " );
        }
        System.out .print("\n\n" );
        ////////////////////////////////////////////////////////////
        System.out .println("After heap sorting : " );
        heapSort.heapSort();
        for (int i = 0; i < scores.length ; i++ ) {
            System.out .print(scores[i] + ", " );
        }
    }
}
```

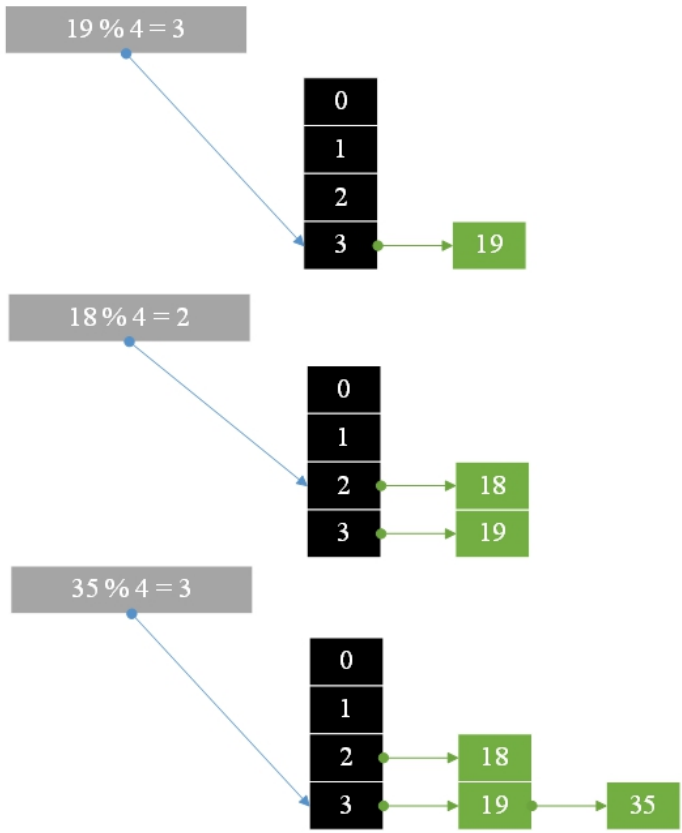
Result:

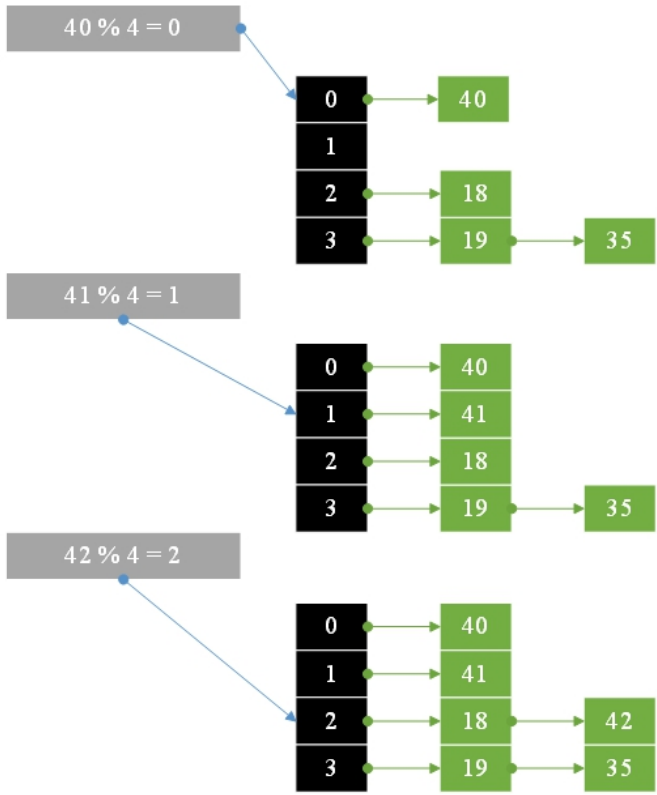

Markers
Properties
Servers
Data Sour...
Snippets
Console
Search
<terminated> TestHeapSort (1) [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (2019
Before building a heap :
10, 90, 20, 80, 30, 70, 40, 60, 50,
After building a heap :
90, 80, 70, 60, 30, 20, 40, 10, 50,
After heap sorting :
10, 20, 30, 40, 50, 60, 70, 80, 90,

Hash Table

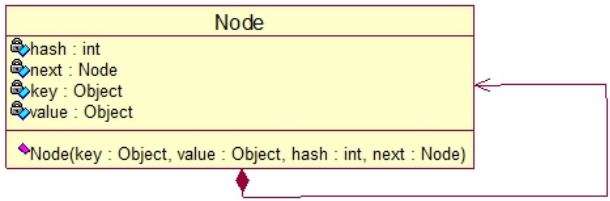
Hash Table:
Access by mapping key = > values in the table.

1. Map {19, 18, 35,40,41,42} to the HashTable mapping rule key % 4




40 % 4 = 0
0
1
2
3
40
18
19
35
41 % 4 = 1
0
1
2
3
40
41
18
19
35
42 % 4 = 2
0
1
2
3
40
41
18
19
42
35

2. Implement a Hashtable



Node.java

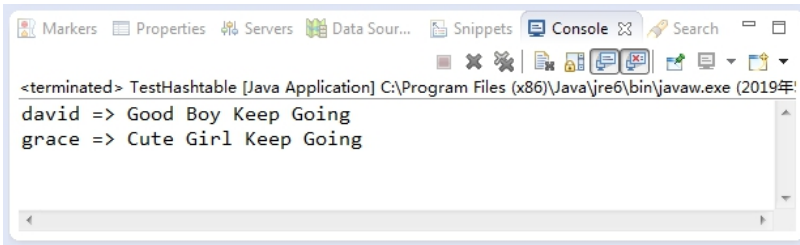
```
public class Node {
    private Object key ;
    private Object value ;
    private int hash ;
    public Node next ;
    public Node(Object key, Object value, int hash, Node next) {
        this .key = key;
        this .value = value;
        this .hash = hash;
        this .next = next;
    }
    public Object getKey() {
        return key ;
    }
    public void setKey(String key) {
        this .key = key;
    }
    public Object getValue() {
        return value ;
    }
    public void setValue(String value) {
        this .value = value;
    }
    public int getHash() {
        return hash ;
    }
    public void setHash(int hash) {
        this .hash = hash;
    }
}
```

Hashtable.java

TestHashtable.java

```
public class TestHashtable {
    public static void main(String[] args) {
        Hashtable table = new Hashtable();
        table.put("david" , "Good Boy Keep Going" );
        table.put("grace" , "Cute Girl Keep Going" );
        System.out .println("david => " +table.get("david" ));
        System.out .println("grace => " +table.get("grace" ));
    }
}
```

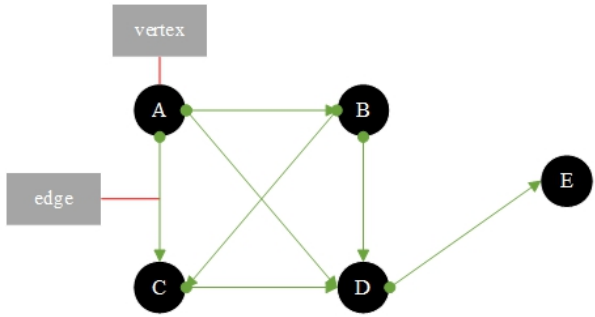
Result:



Directed Graph and Depth-First Search

Directed Graph:
The data structure is represented by an adjacency matrix (that is, a two-dimensional array) and an adjacency list. Each node is called a vertex, and two adjacent nodes are called edges.

Directed Graph has direction : A -> B and B -> A are different

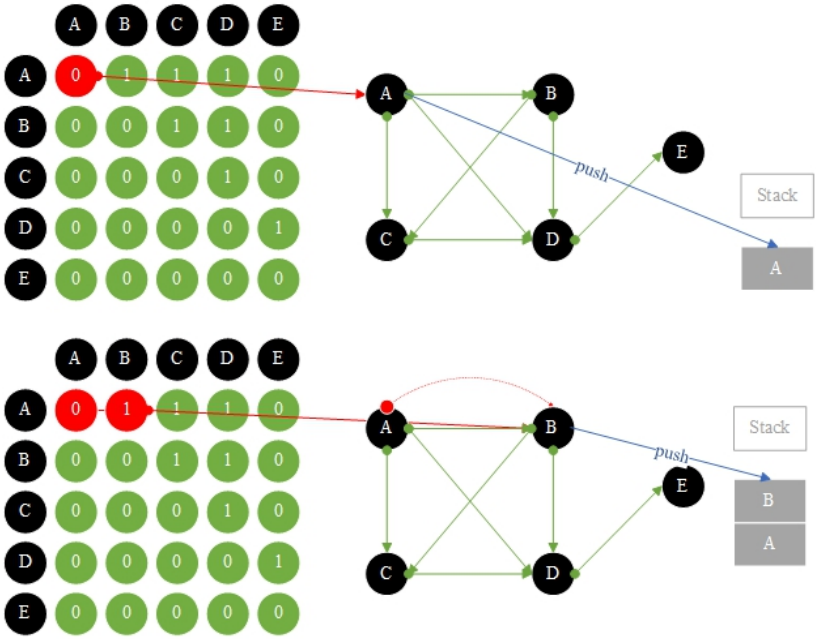


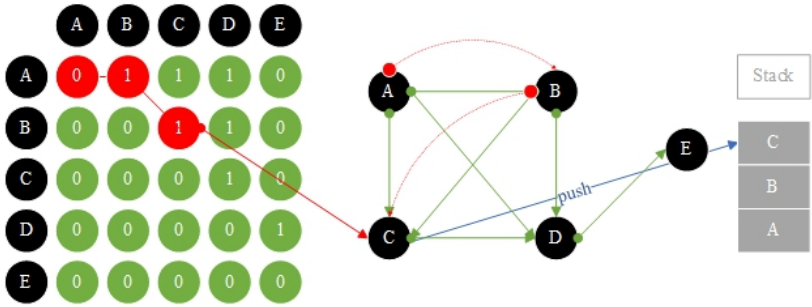
1. The adjacency matrix is described above:
The total number of vertices is a two-dimensional array size, if have value of the edge is 1 , otherwise no value of the edge is 0 .

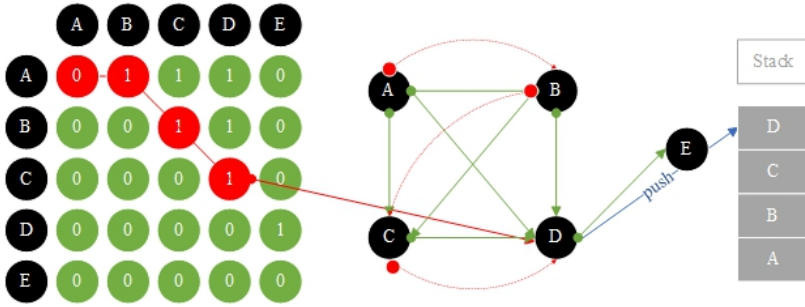
	A	B	C	D	E
A	0	1	1	1	0
B	0	0	1	1	0
C	0	0	0	1	0
D	0	0	0	0	1
E	0	0	0	0	0

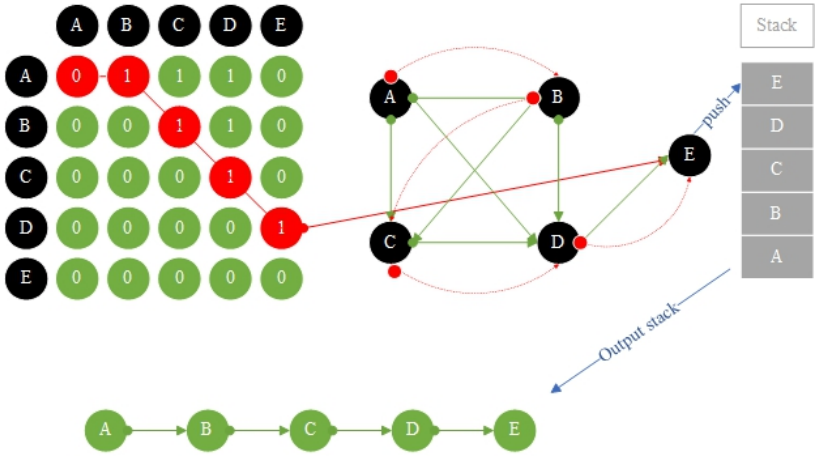
2. Depth-First Search:

Look for the neighboring edge node B from A and then find the neighboring node C from B and so on until all nodes are found A -> B -> C -> D -> E .




A B C D E
A 0 1 1 1 0
B 0 0 1 1 0
C 0 0 0 1 0
D 0 0 0 0 1
E 0 0 0 0 0
A
B
C
D
E
push
Stack
C
B
A


A B C D E
A 0 1 1 1 0
B 0 0 1 1 0
C 0 0 0 1 0
D 0 0 0 0 1
E 0 0 0 0 0
A
B
C
D
E
push
Stack
D
C
B
A


A B C D E
A 0 1 1 1 0
B 0 0 1 1 0
C 0 0 0 1 0
D 0 0 0 0 1
E 0 0 0 0 0
A
B
C
D
E
Stack
E
D
C
B
A
push
Output stack
A B C D E

Vertex.java

```
public class Vertex {
    private String data ;
    private boolean visited ; // Have you visited
    public Vertex() {
        super ();
    }
    public Vertex(String data, boolean visited) {
        this .data = data;
        this .visited = visited;
    }
    public String getData() {
        return data ;
    }
    public void setData(String data) {
        this .data = data;
    }
    public boolean isVisited() {
        return visited ;
    }
    public void setVisited(boolean visited) {
        this .visited = visited;
    }
}
```

Graph.java

```
import java.util.Stack;
public class Graph {
    private int maxVertexSize ; // Two-dimensional array size
    private int size ; // Current vertex size
    private Vertex[] vertexs ;
    private int [][] adjacencyMatrix ;
    private Stack<Integer> stack ;// Stack saves current vertices
    public Graph(int maxVertexSize) {
        this .maxVertexSize = maxVertexSize;
        vertexs = new Vertex[maxVertexSize];
        adjacencyMatrix = new int [maxVertexSize]
[maxVertexSize];
        stack = new Stack<Integer>();
    }
    public void addVertex(String data) {
        Vertex vertex = new Vertex(data, false );
        vertexs [size ] = vertex;
        size ++;
    }
    // Add adjacent edges
    public void addEdge(int from, int to) {
        // A -> B  and B -> A are different
        adjacencyMatrix [from][to] = 1;
    }
    public void depthFirstSearch() {
        // Start searching from the first vertex
        Vertex firstVertex = vertexs [0];
        firstVertex.setVisited(true );
        System.out .print(firstVertex.getData());
        stack .push(0);
        while (!stack .isEmpty()) {
            int row = stack .peek();
            // Get adjacent vertex positions that have not been
visited
            int col = findAdjacencyUnVisitedVertex(row);
            if (col == -1) {
                stack .pop();
            } else {
```

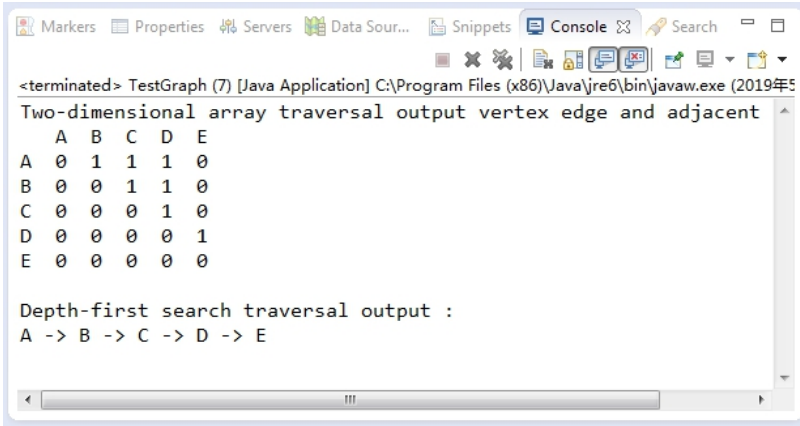
```
                vertexs [col].setVisited(true );
                System.out .print(" -> " +vertexs [col].getData());
                stack .push(col);
            }
        }
        clear();
    }
    // Get adjacent vertex positions that have not been visited
    public int findAdjacencyUnVisitedVertex(int row) {
        for (int col = 0; col < size ; col++) {
            if (adjacencyMatrix [row][col] == 1 && !vertexs
[col].isVisited()) {
                return col;
            }
        }
        return -1;
    }
    // Clear reset
    public void clear() {
        for (int i = 0; i < size ; i++) {
            vertexs [i].setVisited(false );
        }
    }
    public int [][] getAdjacencyMatrix() {
        return adjacencyMatrix ;
    }
    public Vertex[] getVertexs() {
        return vertexs ;
    }
}
```

TestGraph.java

```
public class TestGraph {
    public static void main(String[] args) {
        Graph graph = new Graph(5);
        graph.addVertex("A" );
        graph.addVertex("B" );
        graph.addVertex("C" );
        graph.addVertex("D" );
        graph.addVertex("E" );
```

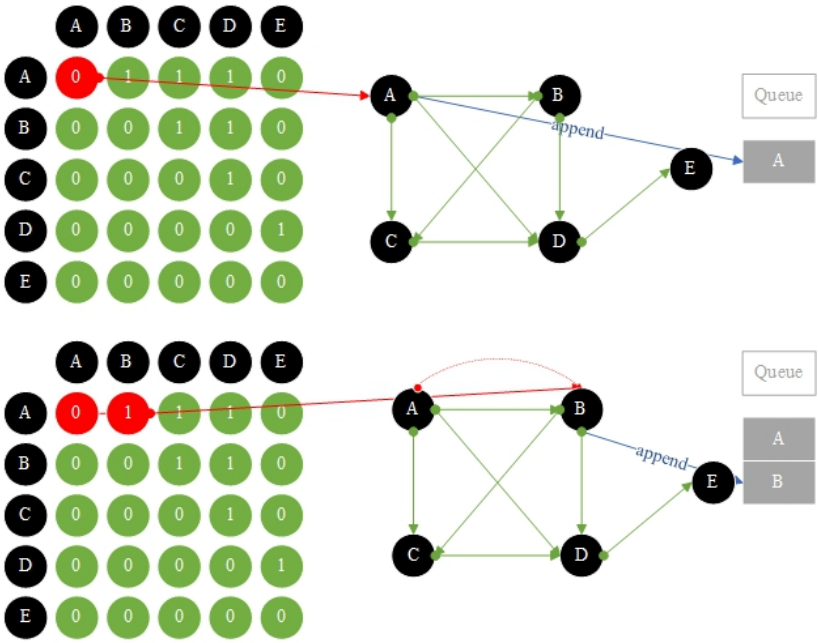
```
        graph.addEdge(0, 1);
        graph.addEdge(0, 2);
        graph.addEdge(0, 3);
        graph.addEdge(1, 2);
        graph.addEdge(1, 3);
        graph.addEdge(2, 3);
        graph.addEdge(3, 4);
        // Two-dimensional array traversal output vertex edge and
adjacent array
        printGraph (graph);
        System.out .print("\nDepth-first search traversal output : \n"
);
        graph.depthFirstSearch();
    }
    public static void printGraph(Graph graph) {
        System.out .print("output vertex edge and adjacent array
: \n  " );
        for (int i = 0; i < graph.getVertexs().length ; i++) {
            System.out .print(graph.getVertexs()[i].getData() + "  " );
        }
        System.out .print("\n" );
        for (int i = 0; i < graph.getAdjacencyMatrix().length ; i+
+) {
            System.out .print(graph.getVertexs()[i].getData() + "  " );
            for (int j = 0; j < graph.getAdjacencyMatrix().length ; j
++) {
                System.out .print(graph.getAdjacencyMatrix()[i][j] +
"  " );
            }
            System.out .print("\n" );
        }
    }
}
```

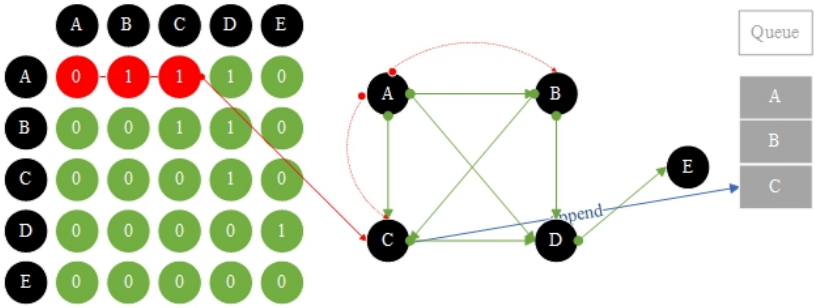
Result:

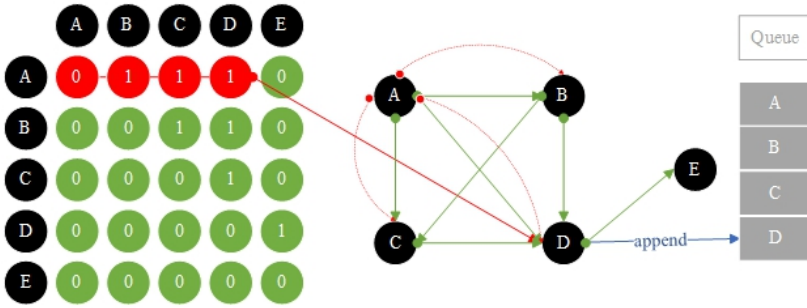

Markers
Properties
Servers
Data Sour...
Snippets
Console
Search
<terminated> TestGraph (7) [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (2019年5
Two-dimensional array traversal output vertex edge and adjacent
A B C D E
A 0 1 1 1 0
B 0 0 1 1 0
C 0 0 0 1 0
D 0 0 0 0 1
E 0 0 0 0 0
Depth-first search traversal output :
A -> B -> C -> D -> E

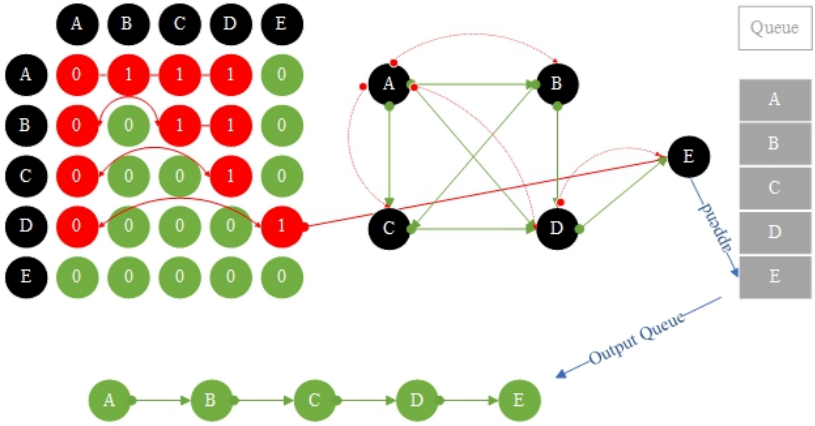
Directed Graph and Breadth-First Search

Breadth-First Search:
Find all neighboring edge nodes B, C, D from A and then find all neighboring nodes A, C, D from B and so on until all nodes are found A -> B -> C -> D -> E .




A B C D E
A 0 1 1 1 0
B 0 0 1 1 0
C 0 0 0 1 0
D 0 0 0 0 1
E 0 0 0 0 0
A
B
C
D
E
append
Queue
A
B
C


A B C D E
A 0 1 1 1 0
B 0 0 1 1 0
C 0 0 0 1 0
D 0 0 0 0 1
E 0 0 0 0 0
A
B
C
D
E
append
Queue
A
B
C
D


A B C D E
A 0 1 1 1 0
B 0 0 1 1 0
C 0 0 0 1 0
D 0 0 0 0 1
E 0 0 0 0 0
A
B
C
D
E
Queue
A
B
C
D
E
append
Output Queue
A
B
C
D
E

Vertex.java

```
public class Vertex {
    private String data ;
    private boolean visited ; // Have you visited
    public Vertex() {
        super ();
    }
    public Vertex(String data, boolean visited) {
        this .data = data;
        this .visited = visited;
    }
    public String getData() {
        return data ;
    }
    public void setData(String data) {
        this .data = data;
    }
    public boolean isVisited() {
        return visited ;
    }
    public void setVisited(boolean visited) {
        this .visited = visited;
    }
}
```

Graph.java

```
import java.util.LinkedList;
import java.util.Queue;
public class Graph {
    private int maxVertexSize ; // Two-dimensional array size
    private int size ; // Current vertex size
    private Vertex[] vertexs ;
    private int [][] adjacencyMatrix ;
    private Queue<Integer> queue ;// Queue saves current
vertices
    public Graph(int maxVertexSize) {
        this .maxVertexSize = maxVertexSize;
        vertexs = new Vertex[maxVertexSize];
        adjacencyMatrix = new int [maxVertexSize]
[maxVertexSize];
        queue = new LinkedList<Integer>();
    }
    public void addVertex(String data) {
        Vertex vertex = new Vertex(data, false );
        vertexs [size ] = vertex;
        size ++;
    }
    // Add adjacent edges
    public void addEdge(int from, int to) {
     // A -> B  and B -> A are different
        adjacencyMatrix [from][to] = 1;
    }
    public void breadthFirstSearch() {
        // Start searching from the first vertex
        Vertex firstVertex = vertexs [0];
        firstVertex.setVisited(true );
        System.out .print(firstVertex.getData());
        queue .add(0);
        int col;
        while (!queue .isEmpty()) {
            int head = queue .remove();
         // Get adjacent vertex positions that have not been visited
            col = findAdjacencyUnVisitedVertex(head);
          //Loop through all vertices connected to the current
```

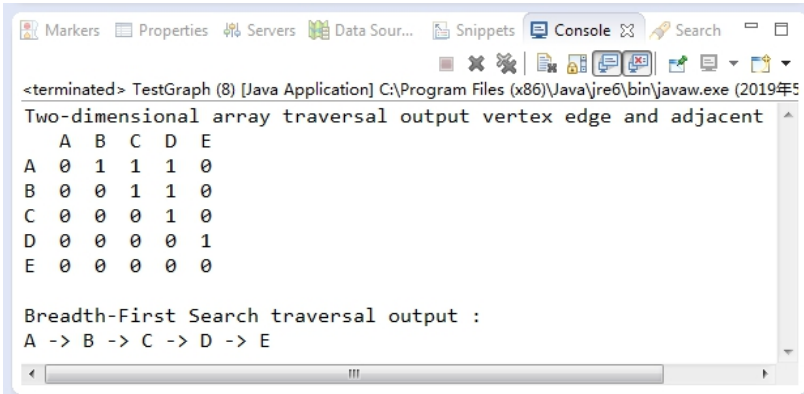
```
vertex
            while (col != -1)
            {
                vertexs [col].setVisited(true );
                System.out .print(" -> " +vertexs [col].getData());
                queue .add(col);
                col = findAdjacencyUnVisitedVertex(head);
            }
        }
        clear();
    }
    // Get adjacent vertex positions that have not been visited
    public int findAdjacencyUnVisitedVertex(int row) {
        for (int col = 0; col < size ; col++) {
            if (adjacencyMatrix [row][col] == 1 && !vertexs
[col].isVisited()) {
                return col;
            }
        }
        return -1;
    }
    // Clear reset
    public void clear() {
        for (int i = 0; i < size ; i++) {
            vertexs [i].setVisited(false );
        }
    }
    public int [][] getAdjacencyMatrix() {
        return adjacencyMatrix ;
    }
    public Vertex[] getVertexs() {
        return vertexs ;
    }
}
```

TestGraph.java

```
public class TestGraph {
    public static void main(String[] args) {
        Graph graph = new Graph(5);
        graph.addVertex("A" );
```

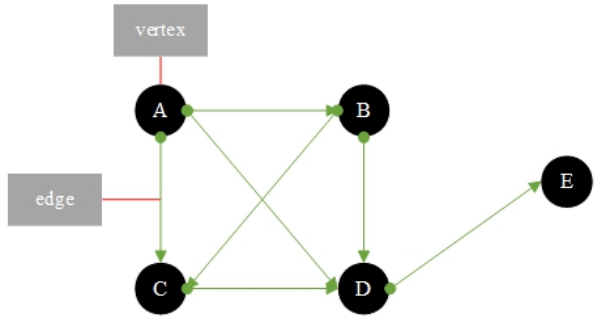
```
        graph.addVertex("B" );
        graph.addVertex("C" );
        graph.addVertex("D" );
        graph.addVertex("E" );
        graph.addEdge(0, 1);
        graph.addEdge(0, 2);
        graph.addEdge(0, 3);
        graph.addEdge(1, 2);
        graph.addEdge(1, 3);
        graph.addEdge(2, 3);
        graph.addEdge(3, 4);
        // Two-dimensional array traversal output vertex edge and
adjacent array
        printGraph (graph);
        System.out .print("\nBreadth-First Search traversal output :
\n" );
        graph.breadthFirstSearch();
    }
    public static void printGraph(Graph graph) {
        System.out .print("output vertex edge and adjacent array
: \n  " );
        for (int i = 0; i < graph.getVertexs().length ; i++) {
            System.out .print(graph.getVertexs()[i].getData() + "  " );
        }
        System.out .print("\n" );
        for (int i = 0; i < graph.getAdjacencyMatrix().length ; i+
+) {
            System.out .print(graph.getVertexs()[i].getData() + "  " );
            for (int j = 0; j < graph.getAdjacencyMatrix().length ; j
++) {
                System.out .print(graph.getAdjacencyMatrix()[i][j] +
"  " );
            }
            System.out .print("\n" );
        }
    }
}
```

Result:

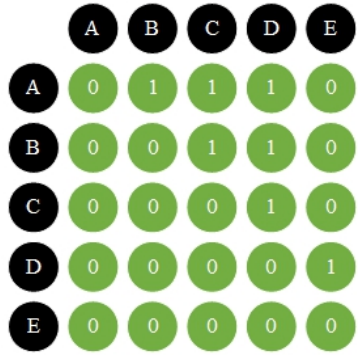

Markers
Properties
Servers
Data Sour...
Snippets
Console
Search
<terminated> TestGraph (8) [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (2019年5
Two-dimensional array traversal output vertex edge and adjacent
A B C D E
A 0 1 1 1 0
B 0 0 1 1 0
C 0 0 0 1 0
D 0 0 0 0 1
E 0 0 0 0 0
Breadth-First Search traversal output :
A -> B -> C -> D -> E

Directed Graph Topological Sorting

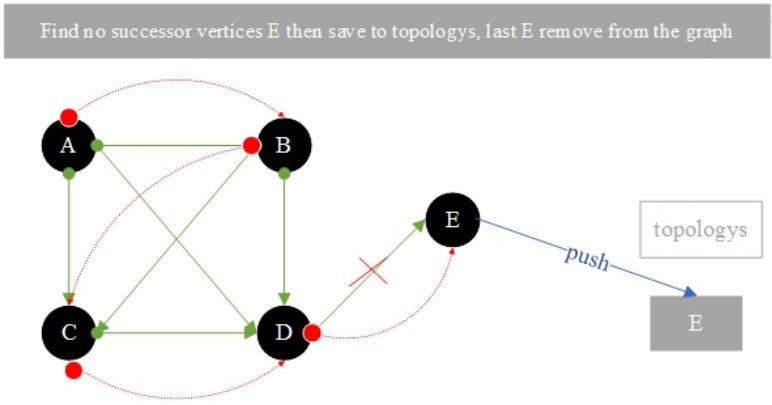
Directed Graph Topological Sorting:
Sort the vertices in the directed graph with order of direction
.
Directed Graph has direction : A -> B and B -> A are different

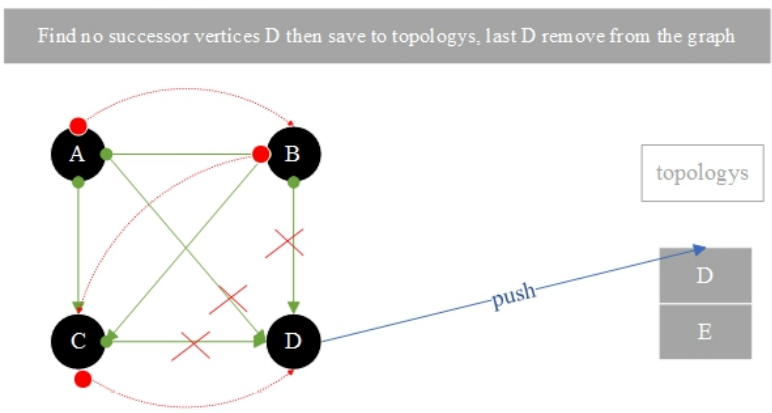


1. The adjacency matrix is described above:
The total number of vertices is a two-dimensional array size, if have value of the edge is 1 , otherwise no value of the edge is 0 .

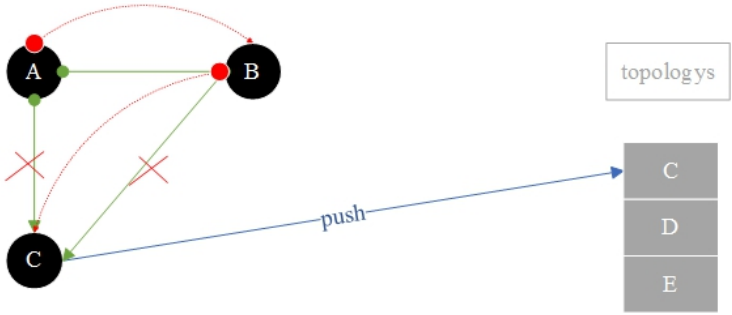


Topological sorting from vertex A : A -> B -> C -> D -> E

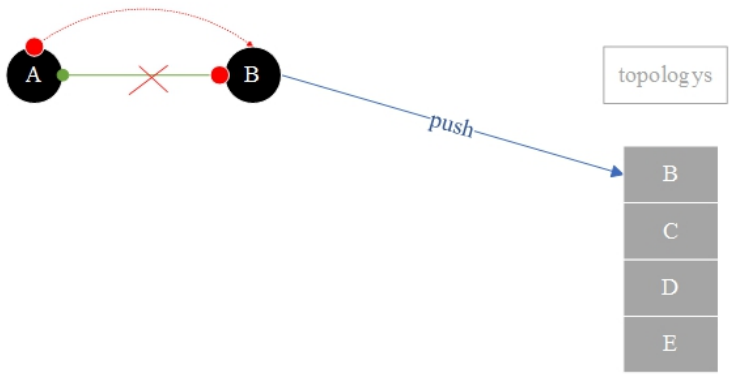




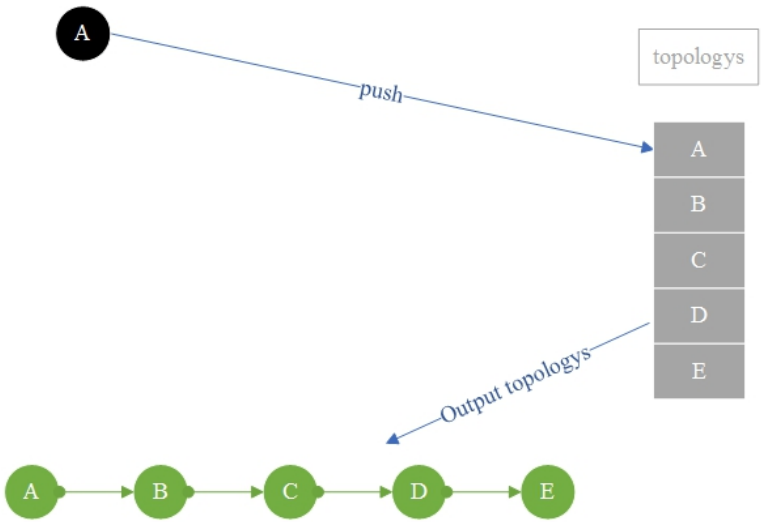
Find no successor vertices C then save to topologys, last C remove from the graph



Find no successor vertices C then save to topologys, last C remove from the graph



Find no successor vertices C then save to topologys, last C remove from the graph



Vertex.java

```
public class Vertex {
    private String data ;
    private boolean visited ; // Have you visited
    public Vertex() {
        super ();
    }
    public Vertex(String data, boolean visited) {
        this .data = data;
        this .visited = visited;
    }
    public String getData() {
        return data ;
    }
    public void setData(String data) {
        this .data = data;
    }
    public boolean isVisited() {
        return visited ;
    }
    public void setVisited(boolean visited) {
        this .visited = visited;
    }
}
```

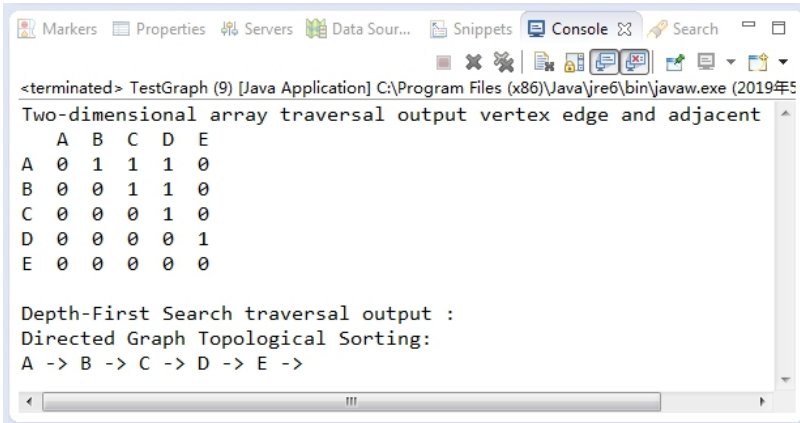
Graph.java

TestGraph.java

```
public class TestGraph {
    public static void main(String[] args) {
        Graph graph = new Graph(5);
        graph.addVertex("A" );
        graph.addVertex("B" );
        graph.addVertex("C" );
        graph.addVertex("D" );
        graph.addVertex("E" );
        graph.addEdge(0, 1);
        graph.addEdge(0, 2);
        graph.addEdge(0, 3);
        graph.addEdge(1, 2);
        graph.addEdge(1, 3);
        graph.addEdge(2, 3);
        graph.addEdge(3, 4);
        // Two-dimensional array traversal output vertex edge and
adjacent array
        printGraph (graph);
        System.out .print("\nDepth-First Search traversal output :
\n" );
        System.out .println("Directed Graph Topological Sorting:" );
        graph.topologySort();
        for (int i=0;i< graph.getTopologys().length ; i++){
            System.out .print(graph.getTopologys()[i].getData()+" -
> " );
        }
    }
    public static void printGraph(Graph graph) {
        System.out .print("traversal output vertex edge and adjacent
array : \n " );
        for (int i = 0; i < graph.getVertexs().length ; i++) {
            System.out .print(graph.getVertexs()[i].getData() + " " );
        }
        System.out .print("\n" );
        for (int i = 0; i < graph.getAdjacencyMatrix().length ; i+
+) {
            System.out .print(graph.getVertexs()[i].getData() + " " );
            for (int j = 0; j < graph.getAdjacencyMatrix().length ; j
```

```
++) {
                System.out .print(graph.getAdjacencyMatrix()[i][j] +
" " );
            }
            System.out .print("\n" );
        }
    }
}
```

Result:


Markers
Properties
Servers
Data Sour...
Snippets
Console
Search
<terminated> TestGraph (9) [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (2019年5
Two-dimensional array traversal output vertex edge and adjacent
A B C D E
A 0 1 1 1 0
B 0 0 1 1 0
C 0 0 0 1 0
D 0 0 0 0 1
E 0 0 0 0 0
Depth-First Search traversal output :
Directed Graph Topological Sorting:
A -> B -> C -> D -> E ->

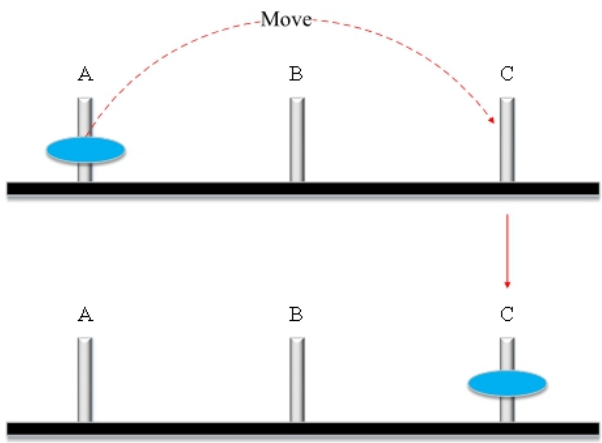
Towers of Hanoi

The Hanoi Tower that a Frenchman M. Claus (Lucas) from Thailand to France in 1883. Hanoi Tower which is supported by three diamond Pillars. At the beginning, God placed 64 gold discs from top to bottom on the first Pillar. God ordered the monks to move all gold discs from the first Pillar to the third Pillar. The principle of large plates under small plates during the handling process. If only one plate is moved daily, the tower will be destroyed. when all the discs are moved that is the end of the world.

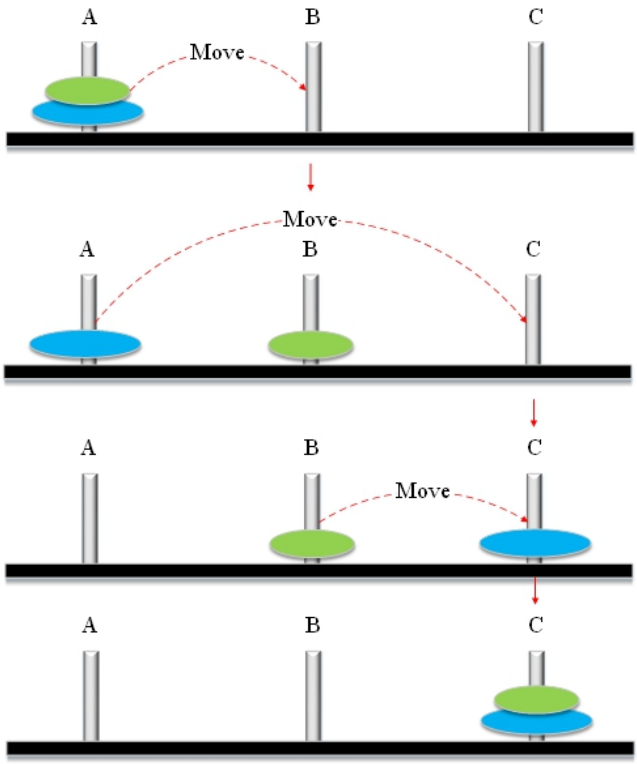
Let's turn this story into an algorithm:

Mark the three columns as ABC.
1. If there is only one disc, move it directly to C (A->C).
2. When there are two discs, use B as an auxiliary (A->B, A->C,B->C).
3. If there are more than two discs, use B as an auxiliary(A->B, A->C,B->C), and continue to recursive process.

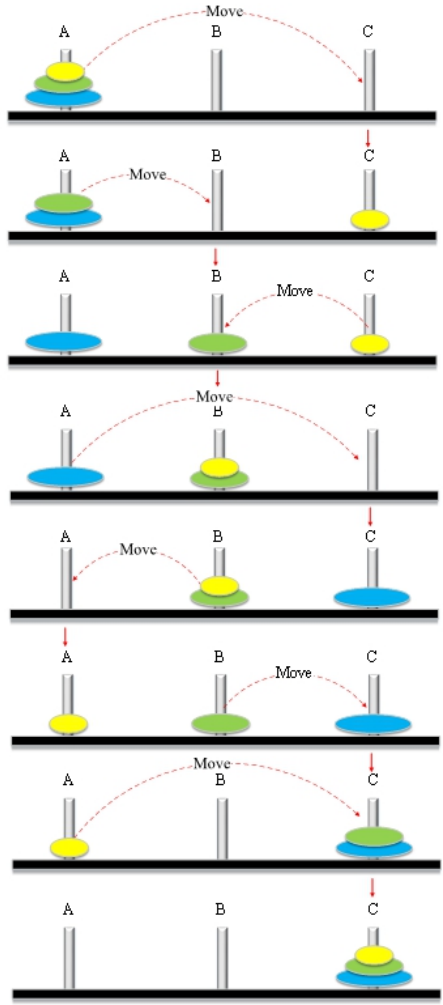
1. If there is only one disc, move it directly to C (A->C).



2. When there are two discs, use B as an auxiliary (A->B, A->C,B->C).



3. If more than two discs, use B as an auxiliary, and continue to recursive process.



TowersOfHanoi.java

```
import java.util.Scanner;
public class TowersOfHanoi {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in );
        System.out .println("Please enter the number of discs :" );
        int n = sc.nextInt();
        hanoi (n, 'A' , 'B' , 'C' );
    }
    public static void hanoi(int n, char A, char B, char C) {
        if (n == 1) {
            System.out .println("Move " + n + " " +A + "  to " +
C);
        } else {
            hanoi (n - 1, A, C, B); // Move the n-1th disc on the
A  through C to  B
            System.out .println("Move " + n + " from " + A + "  to "
+ C);
            hanoi (n - 1, B, A, C); //Move the n-1th disc on the
B  through  A to  C
        }
    }
}
```

Result:

Please enter the number of discs :

1

Move 1 A to C

Run Again Result:

Please enter the number of discs :

2

Move 1 A to B

Move 2 from A to C

Move 1 B to C

Run Again Result:

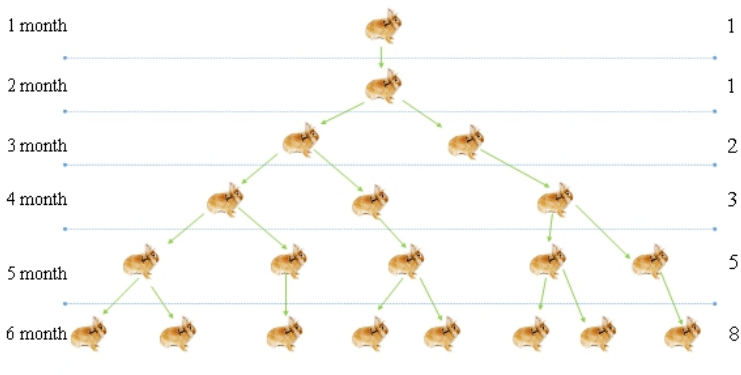
```
Please enter the number of discs :
3
Move 1 A  to C
Move 2 from A  to B
Move 1 C  to B
Move 3 from A  to C
Move 1 B  to A
Move 2 from B  to C
Move 1 A  to C
```

Fibonacci

Fibonacci : a European mathematician in the 1200s, in his writings: "If there is a rabbit
After a month can birth to a newborn rabbit. At first there was only 1 rabbit, after one month still 1 rabbit. after two month 2 rabbit, and after three months there are 3 rabbit .....
for example: 1, 1, 2, 3, 5, 8, 13 ...

Fibonacci definition:

if n = 0, 1
    fn = n
if n > 1
    fn = fn-1 + fn-2



Fibonacci.java

```
import java.util.Scanner;
public class Fibonacci {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in );
        System.out .println("Please enter the number of month :" );
        int number = sc.nextInt();
        for (int i = 1; i <= number; i++) {
            System.out .println(i + " month : " + fibonacci (i));
        }
    }
    public static int fibonacci(int n) {
        if (n == 1 || n == 2) {
            return 1;
        } else {
            return fibonacci (n - 1) + fibonacci (n - 2);
        }
    }
}
```

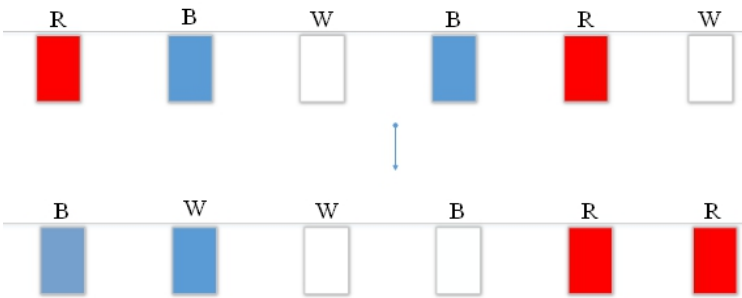
Result:

```
Please enter the number of month :
7
1 month :  1
2 month :  1
3 month :  2
4 month :  3
5 month :  5
6 month :  8
7 month :  13
```

Dijkstra

The tricolor flag was originally raised by E.W. Dijkstra, who used the Dutch national flag (Dijkstra is Dutch).

Suppose there is a rope with red, white, and blue flags. At first all the flags on the rope are not in order. You need to arrange them in the order of blue -> white -> red. How to move them with the least times. you just only do this on the rope, and only swap two flags at a time.



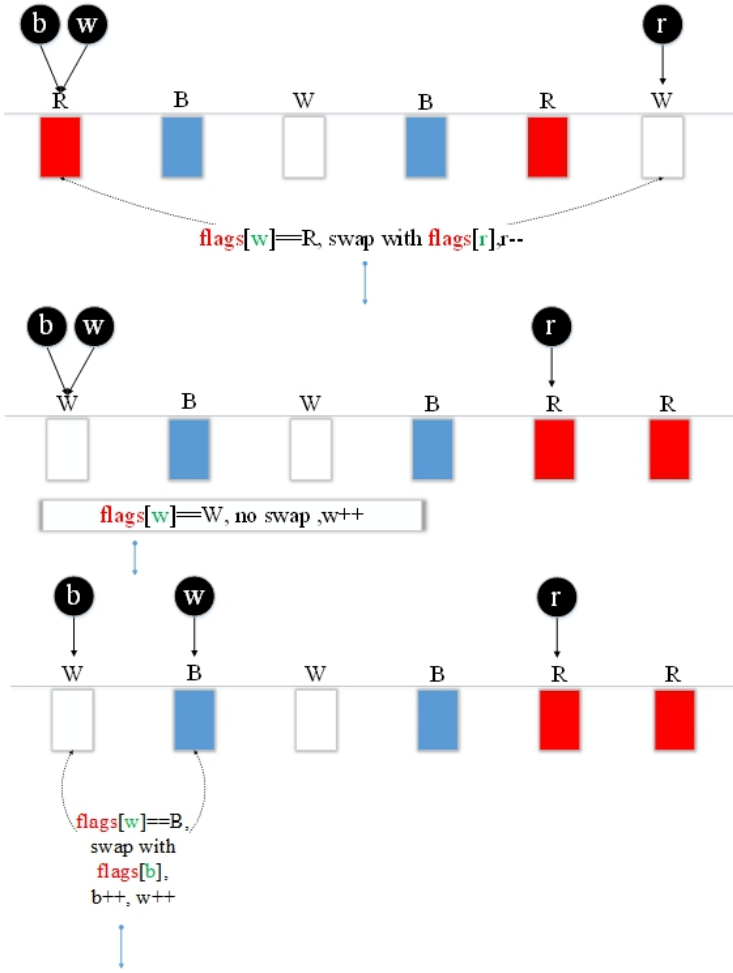
Solution:

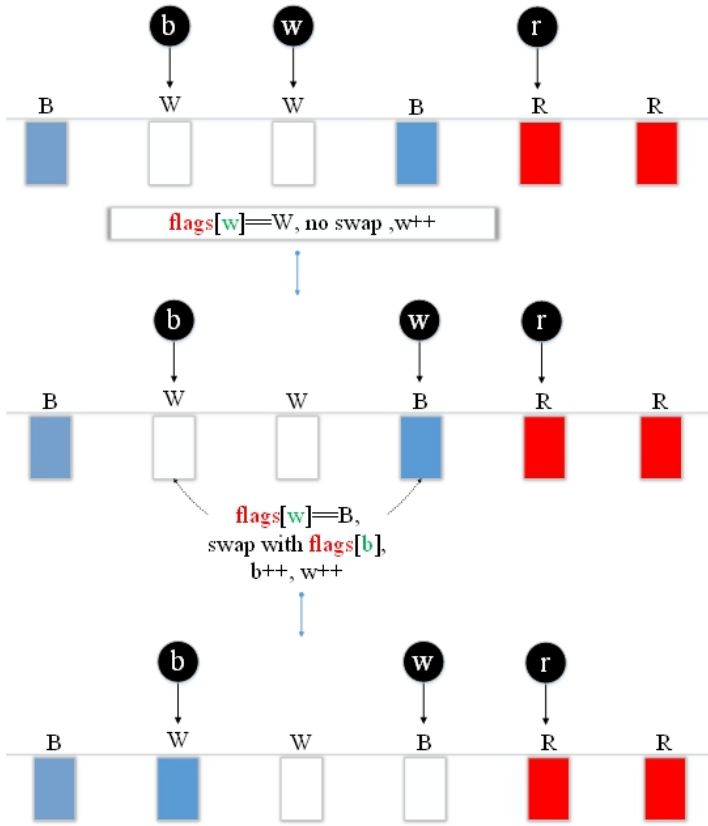
Use the char arrays to store the flags . For example, b , w , and r indicate the position of **blue** , **white** and **red** flags. The beginning of b and w is 0 of the array, and r is at the end of the array.

(1) If the position of w is a blue flag, flags [w] exchange with flags [b]. And whiteIndex and b is moved backward by 1.
(2) If the position of w is a white flag, w moves backward by 1.
(3) If the position of w is a red flag, flags [w] exchange with flags [r]. r moves forward by 1.

In the end, the flags in front of b are all blue, and the flags behind r are all red.

Graphic Solution




b
w
r
B
W
W
B
R
R
flags[w]==W, no swap ,w++
b
w
r
B
W
W
B
R
R
flags[w]==B,
swap with flags[b],
b++, w++
b
w
r
B
W
W
B
R
R

Dijkstra.java

```
import java.util.Scanner;
public class Dijkstra {
    public static void main(String[] args) {
        char [] flags = { 'R' , 'B' , 'W' , 'B' , 'R' , 'W' };
        int b = 0, w = 0, r = flags.length - 1;
        int count = 0;
        while (w <= r) {
            if (flags[w] == 'W' ) {
                w++;
            } else if (flags[w] == 'B' ) {
                char temp = flags[w];
                flags[w] = flags[b];
                flags[b] = temp;
                w++;
                b++;
                count++;
            } else if (flags[w] == 'R' ) {
                char m = flags[w];
                flags[w] = flags[r];
                flags[r] = m;
                r--;
                count++;
            }
        }
        for (int i = 0; i < flags.length ; i++) {
            System.out .print(flags[i]);
        }
        System.out .print("\nThe total exchange count : " + count);
    }
}
```

Result:

BBWWRR
The total exchange count : 4

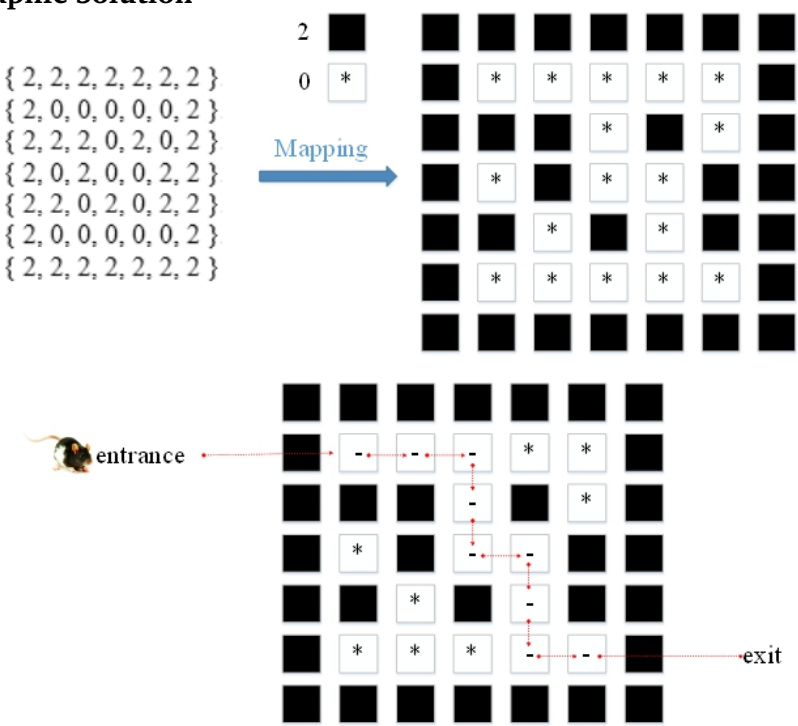
Mouse Walking Maze

Mouse Walking Maze is a basic type of recursive solution. We use 2 to represent the wall in a two-dimensional array , and use 1 to represent the path of the mouse, and try to find the path from the entrance to the exit.

Solution:

The mouse moves in four directions: up, left, down, and right . if hit the wall go back and select the next forward direction, so test the four directions in the array until mouse reach the exit.

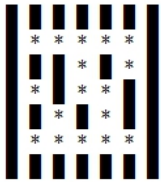
Graphic Solution



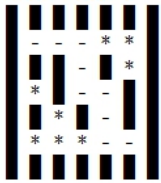
MouseWalkingMaze.java

Result:

```
Maze:
```



```
Maze Path :
```



Eight Coins

There are eight coins with the same appearance, one is a counterfeit coin, and the weight of counterfeit coin is different from the real coin, but it is unknown whether the counterfeit coin is lighter or heavier than the real coin. Please design a efficient algorithm to detect this counterfeit coin.

Solution:

Take six a, b, c, d, e, f from eight coins, and put three to the balance for comparison. Suppose a, b, c are placed on one side, and d, e, f are placed on the other side.

1. $a + b + c > d + e + f$
2. $a + b + c = d + e + f$
3. $a + b + c < d + e + f$

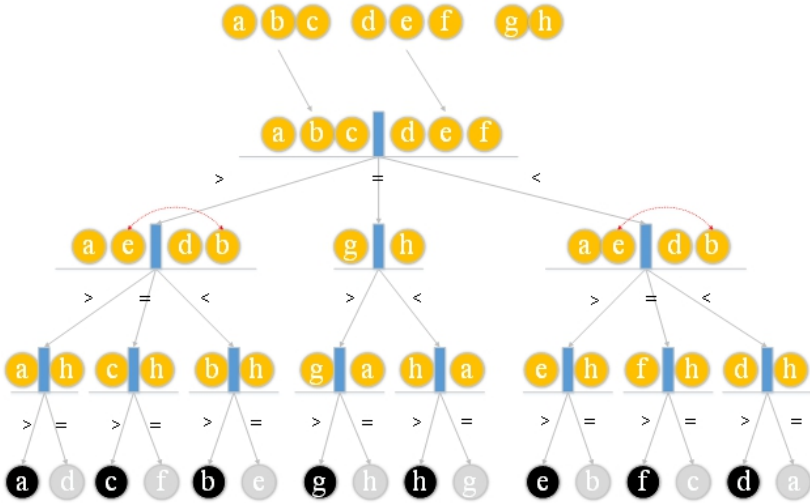
If $a + b + c > d + e + f$, there is a counterfeit coin in one of the six coins, and g, h are real coins. At this time, one coin can be removed from both side. Suppose that c and f are removed. At the same time, one coin at each side is replaced. Suppose the coins b and e are interchanged, and then the second comparison. There are also three possibilities:

1. $a + e > d + b$: the counterfeit currency must be one of a, d . as long as we compare a real currency h with a , we can find the counterfeit currency. If $a > h$, a is a heavier counterfeit currency; if $a = h$, d is a lighter counterfeit currency.
2. $a + e = d + b$: the counterfeit currency must be one of c, f, and the real coin h is compared with c . If $c > h$, c is a heavier counterfeit currency; if $c = h$, then f is a lighter counterfeit currency.
3. $a + e < d + b$: one of b or e is a counterfeit coin , Also use the real coin h to compare with b, if $b > h$, then b is a heavier counterfeit currency; if $b = h$, then e is a lighter counterfeit currency;

Graphic Solution

Heavy

Light


a b c d e f g h
a b c d e f
> = <
a e d b
g h
a e d b
> = <
> <
> = <
a h c h b h
g a h a
e h f h d h
> = > = > =
> = > = > =
> = > = > =
a d c f b e
g h h g
e b f c d a

EightCoins.java

```
import java.util.Random;
import java.util.Scanner;
public class EightCoins {
    public static void compare(int coins[], int i, int j, int k) {//
coin[k] true,coin[i]>coin[j]
        if (coins[i] > coins[k]) //
coin[i]>coin[j]&&coin[i]>coin[k] ----->coin[i] is a heavy
counterfeit coin
            System.out .println("\nCounterfeit currency " + (i + 1)
+ " is heavier " );
        else //coin[j] is a light counterfeit coin
            System.out .println("\nCounterfeit currency " + (j + 1)
+ " is lighter " );
    }
    public static void eightcoins(int coins[]) {
        if (coins[0] + coins[1] + coins[2] == coins[3] +
coins[4] + coins[5]){ //(a+b+c)==(d+e+f)
            if (coins[6] > coins[7]) //g>h?(g>a?g:a):(h>a?h:a)
                compare (coins, 6, 7, 0);
            else //h>g?(h>a?h:a):(g>a?g:a)
                compare (coins, 7, 6, 0);
        } else if (coins[0] + coins[1] + coins[2] > coins[3] +
coins[4] + coins[5]){ //(a+b+c)>(d+e+f)
            if (coins[0] + coins[3] == coins[1] + coins[4]) //(a
+e)==(d+b)
                compare (coins, 2, 5, 0);
            else if (coins[0] + coins[3] > coins[1] + coins[4]) //(a
+e)>(d+b)
                compare (coins, 0, 4, 1);
            if (coins[0] + coins[3] < coins[1] + coins[4]) //(a
+e)<(d+b)
                compare (coins, 1, 3, 0);
        } else if (coins[0] + coins[1] + coins[2] < coins[3] +
coins[4] + coins[5]) {//(a+b+c)<(d+e+f)
            if (coins[0] + coins[3] == coins[1] + coins[4]) //(a
+e)>(d+b)
                compare (coins, 5, 2, 0);
            else if (coins[0] + coins[3] > coins[1] + coins[4]) //(a
```

```
+e)>(d+b)
            compare (coins, 3, 1, 0);
        if (coins[0] + coins[3] < coins[1] + coins[4]) //(a
+e)<(d+b)
            compare (coins, 4, 0, 1);
    }
  }
  public static void main(String[] args) {
    int coins[] = new int [8];
    // Initial coin weight is 10
    for (int i = 0; i < 8; i++)
      coins[i] = 10;
    System.out .println("Enter weight of counterfeit
currency  (larger or smaller than 10) :" );
    Scanner sc = new Scanner(System.in );
    Random rn = new Random();
    coins[rn.nextInt(8)] = sc.nextInt();
    eightcoins (coins);
    for (int i = 0; i < 8; i++)
      System.out .print(coins[i]+" , " );
  }
}
```

Result:

First run:
Enter weight of counterfeit currency (larger or smaller than 10) :
2

Counterfeit currency 2 is lighter
10 , 2 , 10 , 10 , 10 , 10 , 10 , 10 ,

Run again:
Enter weight of counterfeit currency (larger or smaller than 10) :
13

Counterfeit currency 4 is heavier
10 , 10 , 10 , 13 , 10 , 10 , 10 , 10 ,

Knapsack Problem

Suppose you have a backpack with a weight of up to 8 kg, and you want to the backpack with a total price of Products, suppose the fruit (ID, Name,Price and Weight)

Solution:
To solve the optimization problem we can use Dynamic Programming . In the beginning there is a empty set, every time add an element, find the best solution at this stage, until all elements are added. After entering the set finally can get the best solution.

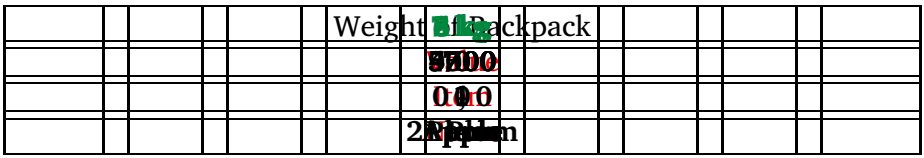
There are two arrays, value and item
value: the total price of the current best solution.
item : the last fruit in the backpack.
there are 8 backpacks with a weight of 1 to 8, and find the best solution for each backpack.

Gradually put the fruit in the backpack and find the best solution:

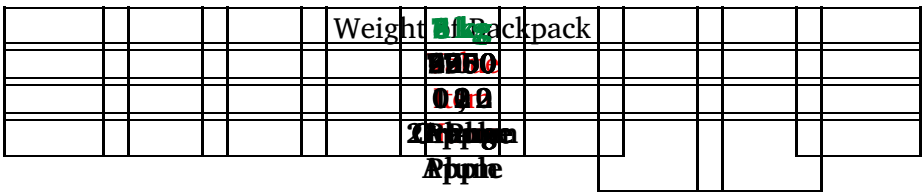
1. **Put in plums:**

						Weight of Backpack										

2. **Put in apples:**



3. **Put in oranges:**



4. **Put in strawberrys:**

5. Put in melons:

From the last table that when the backpack weighs 8 kg, a maximum of 9050, so the best solution is to put Strawberries, Oranges and Apples , and the total price is 9050.

Knapsack.java

```
public class Knapsack {
    public static final int MAXSIZE = 8;
    public static final int MINSIZE = 1;
    public static void main(String[] args) {
        int [] item = new int [MAXSIZE + 1];
        int [] value = new int [MAXSIZE + 1];
        Fruit fruits[] = {
                new Fruit("Plum " , 4, 4500),
                new Fruit("Apple" , 5, 5700),
                new Fruit("Orange" , 2, 2250),
                new Fruit("Strawberry" , 1, 1100),
                new Fruit("Melon" , 6, 6700) };
        for (int i = 0; i < fruits.length ; i++) {
            for (int j = fruits[i].getSize(); j <= MAXSIZE ; j++) {
                int p = j - fruits[i].getSize();
                int newValue = value[p] + fruits[i].getPrice();
                if (newValue > value[j]) { // Find the best solution
                    value[j] = newValue;
                    item[j] = i;
                }
            }
        }
        System.out .println("Item \t Price" );
        for (int i = MAXSIZE ; i >= MINSIZE ; i = i -
fruits[item[i]].getSize()) {
            System.out .println(fruits[item[i]].getName() + "\t" +
fruits[item[i]].getPrice());
        }
        System.out .println("Total \t" + value[MAXSIZE ]);
    }
}
class Fruit {
    private String name ;
    private int size ;
    private int price ;
    public Fruit(String name, int size, int price) {
        this .name = name;
        this .size = size;
```

```
        this .price = price;
    }
    public String getName() {
        return name ;
    }
    public int getPrice() {
        return price ;
    }
    public int getSize() {
        return size ;
    }
}
```

Result:

Item	Price
Strawberry	1100
Orange	2250
Apple	5700
Total	9050

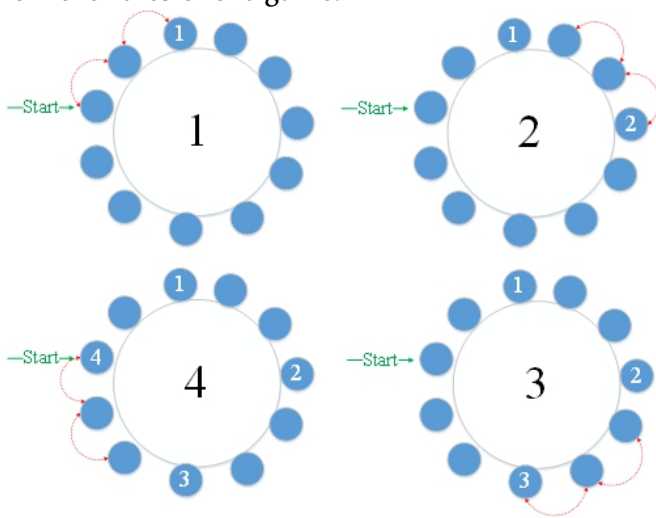
Josephus Problem

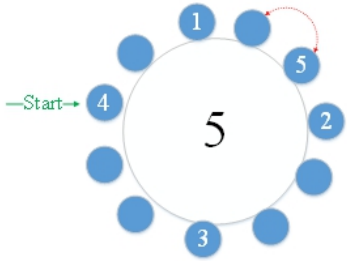
There are 9 Jewish hid in a hole with Josephus and his friends . The 9 Jews decided to die rather than be caught by the enemy, so they decided In a suicide method, 11 people are arranged in a circle, and the first person reports the number. After each number is reported to the third person, the person must commit suicide. Then count again from the next one until everyone commits suicide.But Josephus and his friends did not want to obey. Josephus asked his friends to pretend to obey, and he arranged the friends with himself. In the 2th and 7st positions, they escaped this death game.

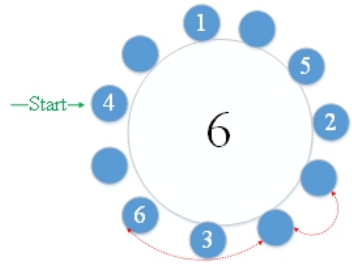
Solution:
As long as the array is treated as a ring. Fill in a count for each dataless area, until the count reaches 11, and then list the array from index 1, you can know that each suicide order in this position is the Joseph's position. The 11-person position is as follows:

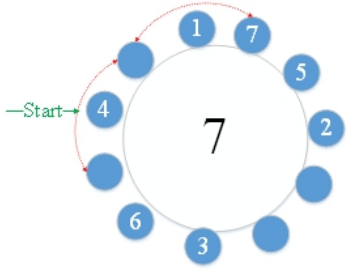
4 10 1 7 5 2 11 9 3 6 8

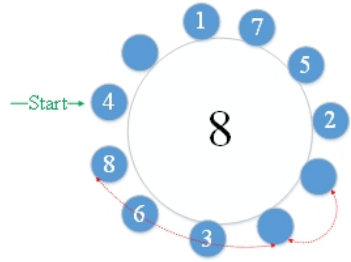
From the above, the last two suicide was in the 31st and 16th position. The previous one
Everyone died, so they didn't know that Joseph and his friends didn't follow the rules of the game.

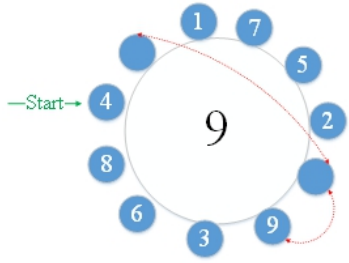


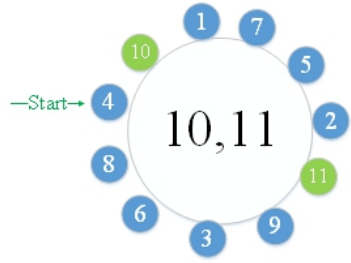

Start
5


Start
6


Start
7


Start
8


Start
9


Start
10,11

Joseph.java

```
import java.util.Scanner;
public class Joseph {
    private static final int N = 11;
    private static final int M = 3;
    public static void main(String[] args) {
        int [] man = new int [N ];
        int count = 1;
        int i = 0, pos = -1;
        int alive = 0;
        while (count <= N ) {
            do {
                pos = (pos + 1) % N ; // Ring
                if (man[pos] == 0)
                    i++;
                if (i == M ) {
                    i = 0;
                    break ;
                }
            } while (true );
            man[pos] = count;
            count++;
        }
        System.out .println("\nJoseph sequenc e :  " );
        for (i = 0; i < N ; i++)
            System.out .print(man[i] + " , " );
    }
}
```

Result:

Joseph sequence :
4 , 10 , 1 , 7 , 5 , 2 , 11 , 9 , 3 , 6 , 8 ,

If you enjoyed this book and found some benefit in reading this, I'd like to hear from you and hope that you could take some time to post a review on Amazon. Your feedback and support will help us to greatly improve in future and make this book even better.

You can follow this link now.

http://www.amazon.com/review/create-review?
&asin=B08BWT6RCT

Different country reviews only need to modify the amazon domain name in the link:
www.amazon.co.uk
www.amazon.de
www.amazon.fr
www.amazon.es
www.amazon.it
www.amazon.ca
www.amazon.nl
www.amazon.in
www.amazon.co.jp
www.amazon.com.br
www.amazon.com.mx
www.amazon.com.au

I wish you all the best in your future success!